GUROBI OPTIMIZER EXAMPLE TOUR

Contents

Introduction

The Gurobi™ distribution includes an extensive set of examples that illustrate commonly used features of the Gurobi libraries. Most examples have versions for C, C++, C#, Java, Visual Basic, and Python. A few, however, illustrate features that are specific to the Python interface.

The distribution also includes examples for our MATLAB® and R interfaces. Note, however, that our interfaces to these languages are built around the assumption that you will use the rich matrix-oriented capabilities of the underlying languages to build your optimization models. Thus, our examples for these languages don't attempt to show you how to build models. We have instead chosen to provide a few simple examples that demonstrate how to pass matrices into our interface.

This document provides a brief tour of these examples. We won't go through each example in detail. Instead, we'll start with an Overview of the set of tasks that you are likely to want to perform with the Gurobi Optimizer. Later sections will then describe how specific examples accomplish each of these tasks. Alternatively, we provide a Structured List of all of our examples, which you can use to dive directly into an example of interest to you. In either case, we suggest that you browse the example source code (in a text editor, or in another browser window) while reading this document. This document includes Source Code for all of the examples, in all available languages. Source files are also available in the `examples` directory of the Gurobi distribution.

If you would like further details on any of the Gurobi routines used in these examples, please consult the Gurobi Reference Manual.

This document provides a quick guided tour of the Gurobi examples; we will try to highlight some of the most important features of these examples. Full source code is provided in this document, so you are free to explore the examples in full detail.

Wherever possible, we try to discuss the examples in a manner that is independent of programming languages. We will refer to each example using a brief, language independent name. You will need to map this name to the specific source file name for your language. For example, the `facility` example corresponds to six different implementations, one in C (facility_c.c), one in C++ (facility_c++.cpp), one in Java (Facility.java), one in C# (facility_cs.cs), one in Visual Basic (facility_vb.vb), and one in Python (facility.py). If you would like to look at the language implementation for a particular example, please refer to the appropriate example source file.

Topics covered in the examples

The easiest place to start your introduction to the Gurobi examples is probably with the examples that load and solve a model from a file. These demonstrate the most basic capabilities of the Gurobi libraries. They also demonstrate the use of model attributes, which are an important concept in the Gurobi optimizer.

Once you are comfortable with these examples, you should move on to the examples that build a model from scratch. These show you how to create variables and constraints, and add them to an optimization model.

The next topic covered in this document is model modification. The Gurobi distribution includes examples that add and remove constraints, add variables, and change variable types, bounds and objective coefficients. You modify a model in much the same way that you build a model from scratch, but there are some important differences involving the use of the solution information.

Next, this document covers parameter changes. The `params` example shows you how to change parameters, and in particular how to use different parameter settings for different models.

On a related note, the tuning section demonstrates the use of our automated tuning tool. This tool searches for parameter settings that improve performance on a particular model.

The infeasibility section considers a few examples that cope with model infeasibility. Some use an Irreducible Inconsistent Subsystem (IIS) to handle the infeasibility, while others relax constraints.

One useful MIP feature that is worth understanding is MIP starts. A MIP start allows you to specify a known feasible solution to the MIP solver. The solution provides a bound on the objective of the best possible solution, which can help to limit the MIP search. The solution also provides a potential start point for the local search heuristics that are utilized by the Gurobi MIP solver.

It is possible to achieve model-data separation when using our Python interface, as is often done in modeling languages, but you need to make use of Python modules to do so. The model-data separation section provides an example of how this is done. It considers three versions of the diet example. All three use the same function to formulate and solve the actual optimization model, but they obtain model data from very different places.

The final topic we cover in this document is Gurobi callbacks. Callbacks allow the user to obtain periodic progress information related to the optimization.

2.1 A list of the Gurobi examples

We recommend that you begin by reading the overview of the examples (which begins in the next section). However, if you'd like to dive directly into a specific example, the following is a list of all of the examples included in the Gurobi distribution, organized by basic function. The source for the examples can be found by following the provided links, or in the `examples` directory of the Gurobi distribution.

Read a model from a file

- **lp** - A very simple example that reads a continuous model from a file, optimizes it, and writes the solution to a file. If the model is infeasible, it writes an Irreducible Inconsistent Subsystem (IIS) instead. C, C++, C#, Java, Python, VB.
- **mip2** - Reads a MIP model from a file, optimizes it, and then solves the fixed version of the MIP model. C, C++, C#, Java, Python, VB.

Build a simple model

- **mip1** - Builds a trivial MIP model, solves it, and prints the solution. C, C++, C#, Java, MATLAB, Python, R, VB.
- **qp** - Builds a trivial QP model, solves it, converts it to an MIQP model, and solves it again. C, C++, C#, Java, MATLAB, Python, R, VB.
- **qcp** - Builds and solves a trivial QCP model. C, C++, C#, Java, MATLAB, Python, R, VB.
- **sos** - Builds and solves a trivial SOS model. C, C++, C#, Java, MATLAB, Python, R, VB.
- **dense** - Solves a model stored using dense matrices. We don't recommend using dense matrices, but this example may be helpful if your data is already in this format. C, C++, C#, Java, Python, VB.
- **genconstr** - Demonstrates the use of general constraints. C, C++, C#, Java, Python, VB.
- **multiobj** - Demonstrates the use of multi-objective optimization. C, C++, C#, Java, Python, VB.
- **piecewise** - Demonstrates the use of piecewise-linear objective functions. C, C++, C#, Java, MATLAB, Python, R, VB.
- **poolsearch** - Demonstrates the use of solution pools. C, C++, C#, Java, Python, VB.

A few simple applications

- **diet** - Builds and solves the classic diet problem. Demonstrates model construction and simple model modification - after the initial model is solved, a constraint is added to limit the number of dairy servings. C, C++, C#, Java, MATLAB, Python, VB.
- **diet2, diet3, diet4, dietmodel** - Python-only variants of the diet example that illustrate model-data separation. diet2.py, diet3.py, diet4.py, dietmodel.py.

- **facility** - Simple facility location model: given a set of plants and a set of warehouses, with transportation costs between them, this example finds the least expensive set of plants to open in order to satisfy product demand. This example demonstrates the use of MIP starts — the example computes an initial, heuristic solution and passes that solution to the MIP solver. C, C++, C#, Java, Python, VB.

- **netflow** - A Python-only example that solves a multi-commodity network flow model. It demonstrates the use of several Python modeling constructs, including dictionaries, tuples, tupledict, and tuplelist objects. Python.

- **portfolio** - A Python-only example that solves a financial portfolio optimization model, where the historical return data is stored using the pandas package and the result is plotted using the matplotlib package. It demonstrates the use of pandas, NumPy, and Matplotlib in conjunction with Gurobi. Python.

- **sudoku** - Reads a Sudoku puzzle dataset from a file, builds a MIP model to solve that model, solves it, and prints the solution. C, C++, C#, Java, Python, VB.

- **workforce1** - Formulates and solves a workforce scheduling model. If the model is infeasible, the example computes and prints an Irreducible Inconsistent Subsystem (IIS). C, C++, C#, Java, Python, VB.

- **workforce2** - An enhancement of `workforce1`. This example solves the same workforce scheduling model, but if the model is infeasible, it computes an IIS, removes one of the associated constraints from the model, and re-solves. This process is repeated until the model becomes feasible. Demonstrates constraint removal. C, C++, C#, Java, Python, VB.

- **workforce3** - A different enhancement of `workforce1`. This example solves the same workforce scheduling model, but if the model is infeasible, it adds artificial variables to each constraint and minimizes the sum of the artificial variables. This corresponds to finding the minimum total change in the right-hand side vector required in order to make the model feasible. Demonstrates variable addition. C, C++, C#, Java, Python, VB.

- **workforce4** - An enhancement of `workforce3`. This example solves the same workforce scheduling model, but it starts with artificial variables in each constraint. It first minimizes the sum of the artificial variables. Then, it introduces a new quadratic objective to balance the workload among the workers. Demonstrates optimization with multiple objective functions. C, C++, C#, Java, Python, VB.

- **workforce5** - An alternative enhancement of `workforce3`. This example solves the same workforce scheduling model, but it starts with artificial variables in each constraint. It formulates a multi-objective model where the primary objective is to minimize the sum of the artificial variables (uncovered shifts), and the secondary objective is to minimize the maximum difference in the number of shifts worked between any pair of workers. Demonstrates multi-objective optimization. C, C++, C#, Java, Python, VB.

Illustrating specific features

- **feasopt** - Reads a MIP model from a file, adds artificial slack variables to relax each constraint, and then minimizes the sum of the artificial variables. It then computes the same relaxation using the *feasibility relaxation* feature. The example demonstrates simple model modification by adding slack variables. It also demonstrates the feasibility relaxation feature. C, C++, C#, Java, Python, VB.

- **lpmethod** - Demonstrates the use of different LP algorithms. Reads a continuous model from a file and solves it using multiple algorithms, reporting which is the quickest for that model. C, C++, C#, Java, Python, VB.

- **lpmod** - Demonstrates the use of advanced starts in LP. Reads a continuous model from a file, solves it, and then modifies one variable bound. The resulting model is then solved in two different ways: starting from the solution of the original model, or restarting from scratch. C, C++, C#, Java, Python, VB.

- **params** - Demonstrates the use of Gurobi parameters. Reads a MIP model from a file, and then spends 5 seconds solving the model with each of four different values of the `MIPFocus` parameter. It compares the optimality gaps for the four different runs, and continues with the `MIPFocus` value that produced the smallest gap. C, C++, C#, Java, Python, VB.

- **sensitivity** - MIP sensitivity analysis. Reads a MIP model, solves it, and then computes the objective impact of fixing each binary variable in the model to 0 or 1. Demonstrates simple MIP model modification by changing variable bounds. C, C++, C#, Java, Python, VB.

- **tune** - Uses the parameter tuning tool to search for improved parameter settings for a model. C, C++, C#, Java, Python, VB.

- **fixanddive** - Implements a simple MIP heuristic. It reads a MIP model from a file, relaxes the integrality conditions, and then solves the relaxation. It then chooses a set of integer variables that take integer or nearly integer values in the relaxation, fixes them to the nearest integer, and solves the relaxation again. This process is repeated until the relaxation is either integer feasible or linearly infeasible. The example demonstrates different types of model modification (relaxing integrality conditions, changing variable bounds, etc.). C, C++, C#, Java, Python, VB.

More advanced features

- **tsp** - Solves a traveling salesman problem using lazy constraints. C, C++, C#, Java, Python, VB.

- **callback** - Demonstrates the use of Gurobi callbacks. C, C++, C#, Java, Python, VB.

2.2 Load and solve a model from a file

Examples: callback, feasopt, fixanddive, lp, lpmethod, lpmod, mip2, params, sensitivity

One of the most basic tasks you can perform with the Gurobi libraries is to read a model from a file, optimize it, and report the result. The `lp` (lp_c.c, lp_c++.cpp, lp_cs.cs, Lp.java, lp.py,

lp_vb.vb) and mip2 (mip2_c.c, mip2_c++.cpp, mip2_cs.cs, Mip2.java, mip2.py, mip2_vb.vb) examples are simple illustratations of how this is done in the various supported Gurobi languages. While the specifics vary from one language to another, the basic structure remains the same for all languages.

After initializing the Gurobi environment, the examples begin by reading the model from the specified file. In C, you call the GRBreadmodel() function:

```
error = GRBreadmodel(masterenv, argv[1], &model);
```

In C++, this is done by constructing a GRBModel object:

```
GRBModel model = GRBModel(env, argv[1]);
```

In C# and Java, this is also done by constructing a GRBModel object:

```
GRBModel model = new GRBModel(env, args[0]);
```

In Python, this is done via the read global function:

```
model = read(sys.argv[1])
```

The next step is to invoke the Gurobi optimizer on the model. In C, you call GRBoptimize() on the model variable:

```
error = GRBoptimize(model);
```

In C++, Java, and Python, this is accomplished by calling the optimize method on the model object:

```
model.optimize();
```

In C#, the first letter of the method name is capitalized:

```
model.Optimize();
```

A successful optimize call populates a set of solution attributes in the model. For example, once the call completes, the X variable attribute contains the solution value for each variable. Similarly, for continuous models, the Pi constraint attribute contains the dual value for each constraint.

The examples then retrieve the value of the model Status attribute to determine the result of the optimization. In the lp example, an optimal solution is written to a solution file (model.sol).

There are many other things you can do once you have read and solved the model. For example, lp checks the solution status — which is highly recommended. If the model is found to be infeasible, this example computes an Irreducible Inconsistent Subsystem (IIS) to isolate the source of the infeasibility.

2.3 Build a model

Examples: diet, facility, genconstr, mip1, multiobj, piecewise, poolsearch, qcp, qp, sos, sudoku, workforce1, workforce2, workforce3, workforce4, workforce5

Several of the Gurobi examples build models from scratch. We start by focusing on two: mip1 and sos. Both build very simple models to illustrate the basic process.

Typically, the first step in building a model is to create an empty model. This is done using the GRBnewmodel function in C:

```
error = GRBnewmodel(env, &model, "mip1", 0, NULL, NULL, NULL, NULL);
```

You can optionally create a set of variables when you create the model, as well as specifying bounds, objective coefficients, and names for these variables. These examples add new variables separately.

In C++, C#, and Java, you create a new model using the `GRBModel` constructor. In Java, this looks like:

```
GRBModel model = new GRBModel(env);
```

In Python, the class is called `Model`, and its constructor is similar to the `GRBModel` constructor for C++ and Java.

Once the model has been created, the typical next step is to add variables. In C, you use the `GRBaddvars` function to add one or more variables:

```
error = GRBaddvars(model, 3, 0, NULL, NULL, NULL, obj, NULL, NULL, vtype, NULL);
```

In C++, Java, and Python, you use the `addVar` method on the `Model` object (`AddVar` in C#). In Java, this looks like:

```
GRBVar x = model.addVar(0.0, 1.0, -1.0, GRB.BINARY, "x");
```

The new variable's lower bound, upper bound, objective coefficient, type, and name are specified as arguments. In C++ and Python, you can omit these arguments and use default values; see the Gurobi Reference Manual for details.

The next step is to add constraints to the model. Linear constraints are added through the `GRBaddconstr` function in C:

```
error = GRBaddconstr(model, 3, ind, val, GRB_LESS_EQUAL, 4.0, "c0");
```

To add a linear constraint in C, you must specify a list of variable indices and coefficients for the left-hand side, a sense for the constraint (e.g., `GRB_LESS_EQUAL`), and a right-hand side constant. You can also give the constraint a name; if you omit the name, Gurobi will assign a default name for the constraint.

In C++, C#, Java, and Python, you build a linear constraint by first building linear expressions for the left- and right-hand sides. In Java, which doesn't support operator overloading, you build an expression as follows:

```
GRBLinExpr expr = new GRBLinExpr();
expr.addTerm(1.0, x); expr.addTerm(2.0, y); expr.addTerm(3.0, z);
```

You then use the `addConstr` method on the `GRBModel` object to add a constraint using these linear expressions for the left- and right-hand sides:

```
model.addConstr(expr, GRB_LESS_EQUAL, 4.0, "c0");
```

For C++, C#, and Python, the standard mathematical operators such as +, *, <= have been overloaded so that the linear expression resembles a traditional mathematical expression. In C++:

```
model.addConstr(x + 2 * y + 3 * z <= 4, "c0");
```

Once the model has been built, the typical next step is to optimize it (using `GRBoptimize` in C, `model.optimize` in C++, Java, and Python, or `model.Optimize` in C#). You can then query the `X` attribute on the variables to retrieve the solution (and the `VarName` attribute to retrieve the variable name for each variable). In C, the `X` attribute is retrieved as follows:

```
error = GRBgetdblattrarray(model, GRB_DBL_ATTR_X, 0, 3, sol);
```

In C++:

```
cout << x.get(GRB_StringAttr_VarName) << " "
     << x.get(GRB_DoubleAttr_X) << endl;
cout << y.get(GRB_StringAttr_VarName) << " "
     << y.get(GRB_DoubleAttr_X) << endl;
cout << z.get(GRB_StringAttr_VarName) << " "
     << z.get(GRB_DoubleAttr_X) << endl;
```

In Java:

```
System.out.println(x.get(GRB.StringAttr.VarName) +
                   " " + x.get(GRB.DoubleAttr.X));
System.out.println(y.get(GRB.StringAttr.VarName) +
                   " " + y.get(GRB.DoubleAttr.X));
System.out.println(z.get(GRB.StringAttr.VarName) +
                   " " + z.get(GRB.DoubleAttr.X));
```

In C#:

```
Console.WriteLine(x.Get(GRB.StringAttr.VarName) +
                  " " + x.Get(GRB.DoubleAttr.X));
Console.WriteLine(y.Get(GRB.StringAttr.VarName) +
                  " " + y.Get(GRB.DoubleAttr.X));
Console.WriteLine(z.Get(GRB.StringAttr.VarName) +
                  " " + z.Get(GRB.DoubleAttr.X));
```

In Python:

```
for v in m.getVars():
  print(v.varName, v.x)
```

When querying or modifying attribute values for an array of constraints or variables, it is generally more efficient to perform the action on the whole array at once. This is quite natural in the C interface, where most of the attribute routines take array arguments. In the C++, C#, Java, and Python interfaces, you can use the `get` and `set` methods on the `GRBModel` object to work directly with arrays of attribute values (`getAttr`/`setAttr` in Python). In the `sudoku` Java example, this is done as follows:

```
double[][][] x = model.get(GRB.DoubleAttr.X, vars);
```

We should point out one important subtlely in our interface. We use a *lazy update* approach to building and modifying a model. When you make changes, they are added to a queue. The queue is only flushed when you optimize the model (or write it to a file). In the uncommon situation where you want to query information about your model before optimizing it, you should call the *update* method before making your query.

2.4 Additional modeling elements

Examples: genconstr, multiobj, piecewise, qcp, qp, sos

A mathematical programming model in its traditional form consists of a linear objective, a set of linear constraints, and a set of continuous or integer decision variables. Gurobi supports a number of additional modeling constructs. In addition to linear constraints, Gurobi also supports SOS constraints, quadratic constraints, and general constraints. In addition to a single linear objective, Gurobi also supports quadratic objectives, piecewise-linear objectives, and multiple linear objectives. Consult the corresponding examples from the Gurobi distribution for simple examples of how to use each of these modeling elements.

2.5 Modify a model

Examples: diet, feasopt, fixanddive, lpmod, sensitivity, workforce3, workforce4, workforce5

This section considers model modification. Modification can take many forms, including adding constraints or variables, deleting constraints or variables, modifying constraint and variable attributes, changing constraint coefficients, etc. The Gurobi examples don't cover all possible modifications, but they cover the most common types.

diet

This example builds a linear model that solves the classic diet problem: to find the minimum cost diet that satisfies a set of daily nutritional requirements. Once the model has been formulated and solved, it adds an additional constraint to limit the number of servings of dairy products and solves the model again. Let's focus on the model modification.

Adding constraints to a model that has already been solved is no different from adding constraints when constructing an initial model. In Python, we can introduce a limit of 6 dairy servings through the following constraint:

```
m.addConstr(buy['milk'] + buy['ice cream'] <= 6, "limit_dairy")
```

For linear models, the previously computed solution can be used as an efficient *warm start* for the modified model. The Gurobi solver retains the previous solution, so the next `optimize` call automatically starts from the previous solution.

lpmod

Changing a variable bound is also straightforward. The `lpmod` example changes a single variable bound, then re-solves the model in two different ways. A variable bound can be changed by modifying the `UB` or `LB` attribute of the variable. In C:

```
error = GRBsetdblattrelement(model, GRB_DBL_ATTR_UB, var, 0.0);
```

In Python:

```
minVar.ub = 0
```

The model is re-solved simply by calling the `optimize` method again. For a continuous model, this starts the optimization from the previous solution. To illustrate the difference when solving the model from an initial, unsolved state, the `lpmod` example calls the `reset` function. In C:

```
error = GRBresetmodel(model);
```

In C++, Java, and Python:

```
m.reset()
```

In C#:

```
m.Reset()
```

When we call the optimize method after resetting the model, optimization starts from scratch. Although the difference in computation time is insignificant for this tiny example, a warm start can make a big difference for larger models.

fixanddive

The `fixanddive` example provides another example of bound modification. In this case, we repeatedly modify a set of variable bounds, utilizing warm starts each time. In C, variables are fixed as follows:

```
for (j = 0; j < nfix; ++j)
{
  fixval = floor(fractional[j].X + 0.5);
  error = GRBsetdblattrelement(model, "LB", fractional[j].index, fixval);
  if (error) goto QUIT;
  error = GRBsetdblattrelement(model, "UB", fractional[j].index, fixval);
  if (error) goto QUIT;
}
```

In Python, they are fixed as follows:

```
for i in range(nfix):
  v = fractional[i]
  fixval = int(v.x + 0.5)
  v.lb = fixval
  v.ub = fixval
```

Again, the subsequent call to `optimize` starts from the previous solution.

sensitivity

The `sensitivity` example computes the amount by which the optimal objective changes if each binary variable is fixed at either 0 or 1. For each binary variable, the example creates and solves a copy of the model with new upper and lower bounds. This example is a MIP, so Gurobi can not make use of advanced start information. As a result, the model is solved from scratch after each bound modification.

feasopt

The last modification example we consider is `feasopt`, which adds variables to existing constraints and also changes the optimization objective. Setting the objective to zero is straightforward: simply call `setObjective` with a zero argument:

```
m.setObjective(0)
```

Adding new variables is somewhat more complex. In the example, we want to add artificial variable(s) to each constraint in order to allow the constraint to be relaxed. We use two artificial variables for equality constraints and one for inequality constraints. The Python code for adding a single artificial variable to constraint `c` is:

```
feasModel.addVar(obj=1.0, name="ArtP_" + c.Constrname, column=Column([1], [c]))
```

We use the `column` argument of the `addVar` method to specify the set of constraints in which the new variable participates, as well as the associated coefficients. In this example, the new variable only participates in the constraint to be relaxed. Default values are used here for all variables attributes except the objective and the variable name.

2.6 Change parameters

Examples: callback, fixanddive, lp, lpmethod, mip2, params, sensitivity

This section illustrates the use of Gurobi parameters. Example `params` reads a MIP model from a file, then solves the model using four different values of the `MIPFocus` parameter, running for five seconds per value (`MIPFocus` chooses the high-level strategy that the MIP solver uses to solve the problem). It then chooses the parameter value that produced the smallest MIP gap, and continues solving the model until it achieves optimality.

The mechanics of setting a parameter are quite simple. To set the `MIPFocus` parameter in C, do the following:

```
GRBsetintparam(GRBgetenv(model), GRB_INT_PAR_MIPFOCUS, i);
```

In C++:

```
model.set(GRB_IntParam_MIPFocus, i);
```

In Java:

```
model.set(GRB.IntParam.MIPFocus, i);
```

In C#:

```
model.Parameters.MIPFocus = 1
```

or

```
model.Set(GRB.IntParam.MIPFocus, i);
```

In Python:

```
model.Params.MIPFocus = i
```

We should add a comment on how parameter settings propagate between different models. When we set the `TimeLimit` parameter on the base model, then make a copy of that model, the parameter setting is carried over to the copy. When we set the `MIPFocus` parameter on the copy, that parameter change has no effect on the other copies, nor on the original model.

2.7 Automated parameter tuning

Example: tune

The next example we consider is `tune`, which demonstrates the use of our automated parameter tuning tool. This tool searches for parameter settings that improve performance on a model. While you would typically invoke the tool through the command line, using our `grbtune` program, it can also be invoked from our APIs. We'll provide only a cursory description of the tool here. We recommend that you consult the *Parameter Tuning Tool* section of the Gurobi Reference Manual for more precise details.

Our tuning example demonstrates a typical use of the tuning tool. You would start by invoking the tool on a model. In C:

```
error = GRBtunemodel(model);
```

In Java:

```
model.tune();
```

This routine solves the model multiple times, with different parameter settings, to find settings that improve performance.

Once tuning is complete, you would then use `GetTuneResult` to retrieve the result. In C:

```
error = GRBgettuneresult(model, 0);
```

In Java:

```
model.getTuneResult(0);
```

The numerical argument indicates which tuning result to retrieve (0 is the best result, 1 is the second-best, etc.). This routine loads the requested parameter set into the environment associated with the argument model.

Once the tune parameter settings have been loaded into the model, you can then call `Optimize` to use these parameters to solve the model, or you can call `Write` to write these parameters to a `.prm` file.

2.8 Diagnose and cope with infeasibility

Examples: feasopt, lp, workforce1, workforce2, workforce3

When solving optimization models, there are some situations where the specified constraints cannot be satisfied. When this happens, you often need to either identify and repair the root cause of the infeasibility, or alternatively find a set of constraints to relax in order to obtain a feasible model. The `workforce1`, `workforce2`, and `workforce3` illustrate these different strategies.

Starting with the simplest of the three examples, `workforce1` formulates a simple workforce scheduling model and solves it. If the model is infeasible, it computes an Irreducible Inconsistent

Subsystem (IIS). The user can then inspect this information to understand and hopefully address the source of the infeasibility in the model.

Example `workforce2` is similar, except that if the model is infeasible, the example repeatedly identifies an IIS and removes one of the associated constraints from the model until the model becomes feasible. Note that it is sufficient to remove one constraint from the IIS to address that source of infeasibility, but that one IIS may not capture all sources of infeasibility. It is therefore necessary to repeat the process until the model is feasible.

Example `workforce3` takes a different approach to addressing infeasibility. Rather than identifying and removing IIS members, it allows the constraints of the model to be relaxed. Like the `feasopt` example, an artificial variable is added to each constraint. The example sets the objective on the original variables to zero, and then solves a model that minimizes the total magnitude of the constraint relaxation.

The `feasopt` example demonstrates another approach to relaxing an infeasible model. It computes a *feasibility relaxation* for the infeasible model. A feasibility relaxation is a model that, when solved, minimizes the amount by which the solution violates the bounds and linear constraints of the original model. This method is invoked as follows:

In C:

```
error = GRBfeasrelax(feasmodel, GRB_FEASRELAX_LINEAR, 1,
                     NULL, NULL, rhspen, &feasobj);
```

In C++:

```
feasmodel1.feasRelax(GRB_FEASRELAX_LINEAR, true, false, true);
```

In C#:

```
feasmodel1.FeasRelax(GRB.FEASRELAX_LINEAR, true, false, true);
```

In Java:

```
feasmodel1.feasRelax(GRB.FEASRELAX_LINEAR, true, false, true);
```

In Python:

```
feasmodel1.FeasRelaxS(0, True, False, True);
```

The arguments to this method select the objective function for the relaxed model, the specific set of bounds and constraints that are allowed to be relaxed, and the penalties for relaxing specific bounds and constraints.

2.9 MIP starts

Example: facility

A MIP modeler often knows how to compute a feasible solution to their problem. In cases where the MIP solver is slow in finding an initial feasible solution, it can helpful for the modeler to provide a feasible solution along with the model itself. This is done through the `Start` attribute on the variables. This is illustrated in the `facility` example.

The `facility` example solves a simple facility location problem. The model contains a set of warehouses, and a set of plants that produce the products required in the warehouses. Each

plant has a maximum production capacity and a fixed operating cost. Additionally, there is a cost associated with shipping products from a plant to a warehouse. The goal is to decide which plants should satisfy the demand for the product, given the associated capacities and costs.

The example uses a simple heuristic for choosing an initial solution: it closes the plant with the highest fixed cost. The associated solution may not be optimal, but it could produce a reasonable starting solution for the MIP optimization. The MIP start is passed to the MIP solver by setting the `Start` attribute before the optimization begins. In C, we set the start attribute to open all plants using the following code:

```
for (p = 0; p < nPlants; ++p)
{
  error = GRBsetdblattrelement(model, "Start", opencol(p), 1.0);
  if (error) goto QUIT;
}
```

In Python:

```
for p in range(nPlants):
  open[p].start = 1.0
```

When you run the example, the MIP solver reports that the start produced a feasible initial solution:

```
Loaded MIP start with objective 210500
```

This initial solution turns out to be optimal for the sample data. Although the computation difference is insignificant for this tiny example, providing a good starting solution can sometimes help for more difficult models.

Note that the MIP start in this example only specifies values for some of the variables – the variables that determine which plants to leave open and which plants to close. The Gurobi MIP solve uses whatever start information is provided to try to construct a complete solution.

2.10 Model-data separation in Python

Examples: diet2.py, diet3.py, diet4.py

When building an optimization model in a modeling language, it is typical to separate the optimization model itself from the data used to create an instance of the model. These two model ingredients are often stored in completely different files. We show how a similar result can be achieved in our Python interface with our `diet2.py`, `diet3.py`, and `diet4.py` examples. These examples illustrate alternate approaches to providing data to the optimization model: `diet2.py` embeds the data in the source file, `diet3.py` reads the data from an SQL database (using the Python `sqlite3` package), and `diet4.py` reads the data from an Excel spreadsheet (using the Python `xlrd` package). `dietmodel.py` contains the optimization model itself. The same model is used by `diet2.py`, `diet3.py`, and `diet4.py`.

The key construct that enables the separation of the model from the data is the Python module. A module is simply a set of functions and variables, stored in a file. You import a module into a program using the `import` statement. `diet2.py`, `diet3.py`, and `diet4.py` all populate a set of variables, and then pass them to the `solve` function of the `dietmodel` module using the following pair of statements:

```
import dietmodel
dietmodel.solve(categories, minNutrition, maxNutrition, foods, cost, nutritionValues)
```

The first statement imports the `dietmodel` module, which must be stored in file `dietmodel.py` in the current directory. The second passes the model data to the `solve` function in the newly imported module.

2.11 Callbacks

Example: callback

The final example we consider is `callback`, which demonstrates the use of Gurobi callbacks. Callbacks are used to report on the progress of the optimization or to modify the behavior of the Gurobi solver. To use a callback, the user writes a routine that implements the desired behavior. The routine is passed to the Gurobi optimizer when optimization begins, and the routine is called regularly during the optimization process. One argument of the user routine is a `where` value, which indicates from where in the optimization process the callback is invoked. The user callback routine can call the optimization library to query certain values. We refer the reader to the callback section of the Gurobi Reference Manual for more precise details.

Our callback example implements a simple termination scheme: the user passes a node count into the callback, and the callback asks the optimizer to terminate when that node count is reached. This is implemented in C as follows:

```
GRBcbget(cbdata, where, GRB_CB_MIP_NODCNT, &nodecnt);
if (nodecnt > limit)
  GRBterminate(model);
```

In Python, this is implemented as:

```
nodecnt = model.cbGet(GRB.Callback.MIP_NODCNT)
if nodecnt > model._mynodelimit:
  model.terminate()
```

To obtain the current node count, the user routine calls the `cbget` routine (the `GRBcbget` function in C, or the `cbGet` method on the model object in C++, C#, Java, and Python).

Our callback example also prints progress information. In C:

```
GRBcbget(cbdata, where, GRB_CB_MIP_NODCNT, &nodecnt);
if (nodecnt - mydata->lastmsg >= 100) {
  ...
  printf("%7.0f ...", nodecnt, ...);
}
```

In Python:

```
nodecnt = model.cbGet(GRB.Callback.MIP_NODCNT)
if nodecnt % 100 == 0:
  print(int(nodecnt), "...")
```

Again, the user callback calls the `cbGet` routine to query the state of the optimization.

We have included source code for all of the distributed examples in this section. The identical example source code is included in the `examples` directory in the Gurobi distribution.

3.1 C Examples

This section includes source code for all of the Gurobi C examples. The same source code can be found in the `examples/c` directory of the Gurobi distribution.

callback_c.c

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/*
   This example reads a model from a file, sets up a callback that
   monitors optimization progress and implements a custom
   termination strategy, and outputs progress information to the
   screen and to a log file.

   The termination strategy implemented in this callback stops the
   optimization of a MIP model once at least one of the following two
   conditions have been satisfied:
     1) The optimality gap is less than 10%
     2) At least 10000 nodes have been explored, and an integer feasible
        solution has been found.
   Note that termination is normally handled through Gurobi parameters
   (MIPGap, NodeLimit, etc.).  You should only use a callback for
   termination if the available parameters don't capture your desired
   termination criterion.
*/

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "gurobi_c.h"

/* Define structure to pass my data to the callback function */

struct callback_data {
  double  lastiter;
  double  lastnode;
```

```
  double *solution;
  FILE   *logfile;
};

/* Define my callback function */

int __stdcall
mycallback(GRBmodel *model,
           void     *cbdata,
           int       where,
           void     *usrdata)
{
  struct callback_data *mydata = (struct callback_data *) usrdata;

  if (where == GRB_CB_POLLING) {
    /* Ignore polling callback */
  } else if (where == GRB_CB_PRESOLVE) {
    /* Presolve callback */
    int cdels, rdels;
    GRBcbget(cbdata, where, GRB_CB_PRE_COLDEL, &cdels);
    GRBcbget(cbdata, where, GRB_CB_PRE_ROWDEL, &rdels);
    if (cdels || rdels) {
      printf("%7d columns and %7d rows are removed\n", cdels, rdels);
    }
  } else if (where == GRB_CB_SIMPLEX) {
    /* Simplex callback */
    double itcnt, obj, pinf, dinf;
    int    ispert;
    char   ch;
    GRBcbget(cbdata, where, GRB_CB_SPX_ITRCNT, &itcnt);
    if (itcnt - mydata->lastiter >= 100) {
      mydata->lastiter = itcnt;
      GRBcbget(cbdata, where, GRB_CB_SPX_OBJVAL, &obj);
      GRBcbget(cbdata, where, GRB_CB_SPX_ISPERT, &ispert);
      GRBcbget(cbdata, where, GRB_CB_SPX_PRIMINF, &pinf);
      GRBcbget(cbdata, where, GRB_CB_SPX_DUALINF, &dinf);
      if      (ispert == 0) ch = ' ';
      else if (ispert == 1) ch = 'S';
      else                  ch = 'P';
      printf("%7.0f %14.7e%c %13.6e %13.6e\n", itcnt, obj, ch, pinf, dinf);
    }
  } else if (where == GRB_CB_MIP) {
    /* General MIP callback */
    double nodecnt, objbst, objbnd, actnodes, itcnt;
    int    solcnt, cutcnt;
```

```
    GRBcbget(cbdata, where, GRB_CB_MIP_NODCNT, &nodecnt);
    GRBcbget(cbdata, where, GRB_CB_MIP_OBJBST, &objbst);
    GRBcbget(cbdata, where, GRB_CB_MIP_OBJBND, &objbnd);
    GRBcbget(cbdata, where, GRB_CB_MIP_SOLCNT, &solcnt);
    if (nodecnt - mydata->lastnode >= 100) {
      mydata->lastnode = nodecnt;
      GRBcbget(cbdata, where, GRB_CB_MIP_NODLFT, &actnodes);
      GRBcbget(cbdata, where, GRB_CB_MIP_ITRCNT, &itcnt);
      GRBcbget(cbdata, where, GRB_CB_MIP_CUTCNT, &cutcnt);
      printf("%7.0f %7.0f %8.0f %13.6e %13.6e %7d %7d\n",
             nodecnt, actnodes, itcnt, objbst, objbnd, solcnt, cutcnt);
    }
    if (fabs(objbst - objbnd) < 0.1 * (1.0 + fabs(objbst))) {
      printf("Stop early - 10%% gap achieved\n");
      GRBterminate(model);
    }
    if (nodecnt >= 10000 && solcnt) {
      printf("Stop early - 10000 nodes explored\n");
      GRBterminate(model);
    }
  } else if (where == GRB_CB_MIPSOL) {
    /* MIP solution callback */
    double nodecnt, obj;
    int    solcnt;
    GRBcbget(cbdata, where, GRB_CB_MIPSOL_NODCNT, &nodecnt);
    GRBcbget(cbdata, where, GRB_CB_MIPSOL_OBJ, &obj);
    GRBcbget(cbdata, where, GRB_CB_MIPSOL_SOLCNT, &solcnt);
    GRBcbget(cbdata, where, GRB_CB_MIPSOL_SOL, mydata->solution);
    printf("**** New solution at node %.0f, obj %g, sol %d, x[0] = %.2f ****\n",
           nodecnt, obj, solcnt, mydata->solution[0]);
  } else if (where == GRB_CB_MIPNODE) {
    int status;
    /* MIP node callback */
    printf("**** New node ****\n");
    GRBcbget(cbdata, where, GRB_CB_MIPNODE_STATUS, &status);
    if (status == GRB_OPTIMAL) {
      GRBcbget(cbdata, where, GRB_CB_MIPNODE_REL, mydata->solution);
      GRBcbsolution(cbdata, mydata->solution, NULL);
    }
  } else if (where == GRB_CB_BARRIER) {
    /* Barrier callback */
    int    itcnt;
    double primobj, dualobj, priminf, dualinf, compl;
    GRBcbget(cbdata, where, GRB_CB_BARRIER_ITRCNT, &itcnt);
    GRBcbget(cbdata, where, GRB_CB_BARRIER_PRIMOBJ, &primobj);
```

```
    GRBcbget(cbdata, where, GRB_CB_BARRIER_DUALOBJ, &dualobj);
    GRBcbget(cbdata, where, GRB_CB_BARRIER_PRIMINF, &priminf);
    GRBcbget(cbdata, where, GRB_CB_BARRIER_DUALINF, &dualinf);
    GRBcbget(cbdata, where, GRB_CB_BARRIER_COMPL, &compl);
    printf("%d %.4e %.4e %.4e %.4e %.4e\n",
           itcnt, primobj, dualobj, priminf, dualinf, compl);
  } else if (where == GRB_CB_MESSAGE) {
    /* Message callback */
    char *msg;
    GRBcbget(cbdata, where, GRB_CB_MSG_STRING, &msg);
    fprintf(mydata->logfile, "%s", msg);
  }
  return 0;
}

int
main(int   argc,
     char *argv[])
{
  GRBenv   *env   = NULL;
  GRBmodel *model = NULL;
  int       error = 0;
  int       numvars, solcount, optimstatus, j;
  double    objval, x;
  char     *varname;
  struct callback_data mydata;

  mydata.lastiter = -GRB_INFINITY;
  mydata.lastnode = -GRB_INFINITY;
  mydata.solution = NULL;
  mydata.logfile  = NULL;

  if (argc < 2) {
    fprintf(stderr, "Usage: callback_c filename\n");
    goto QUIT;
  }

  /* Open log file */
  mydata.logfile = fopen("cb.log", "w");
  if (!mydata.logfile) {
    fprintf(stderr, "Cannot open cb.log for callback message\n");
    goto QUIT;
  }

  /* Create environment */
```

```
error = GRBloadenv(&env, NULL);
if (error) goto QUIT;

/* Turn off display and heuristics */

error = GRBsetintparam(env, GRB_INT_PAR_OUTPUTFLAG, 0);
if (error) goto QUIT;

error = GRBsetdblparam(env, GRB_DBL_PAR_HEURISTICS, 0.0);
if (error) goto QUIT;

/* Read model from file */

error = GRBreadmodel(env, argv[1], &model);
if (error) goto QUIT;

/* Allocate space for solution */

error = GRBgetintattr(model, GRB_INT_ATTR_NUMVARS, &numvars);
if (error) goto QUIT;

mydata.solution = malloc(numvars*sizeof(double));
if (mydata.solution == NULL) {
  fprintf(stderr, "Failed to allocate memory\n");
  exit(1);
}

/* Set callback function */

error = GRBsetcallbackfunc(model, mycallback, (void *) &mydata);
if (error) goto QUIT;

/* Solve model */

error = GRBoptimize(model);
if (error) goto QUIT;

/* Capture solution information */

printf("\nOptimization complete\n");

error = GRBgetintattr(model, GRB_INT_ATTR_SOLCOUNT, &solcount);
if (error) goto QUIT;
```

```
  error = GRBgetintattr(model, GRB_INT_ATTR_STATUS, &optimstatus);
  if (error) goto QUIT;

  if (solcount == 0) {
    printf("No solution found, optimization status = %d\n", optimstatus);
    goto QUIT;
  }

  error = GRBgetdblattr(model, GRB_DBL_ATTR_OBJVAL, &objval);
  if (error) goto QUIT;

  printf("Solution found, objective = %.4e\n", objval);

  for ( j = 0; j < numvars; ++j ) {
    error = GRBgetstrattrelement(model, GRB_STR_ATTR_VARNAME, j, &varname);
    if (error) goto QUIT;
    error = GRBgetdblattrelement(model, GRB_DBL_ATTR_X, j, &x);
    if (error) goto QUIT;
    if (x != 0.0) {
      printf("%s %f\n", varname, x);
    }
  }

QUIT:

  /* Error reporting */

  if (error) {
    printf("ERROR: %s\n", GRBgeterrormsg(env));
    exit(1);
  }

  /* Close log file */

  if (mydata.logfile)
    fclose(mydata.logfile);

  /* Free solution */

  if (mydata.solution)
    free(mydata.solution);

  /* Free model */

  GRBfreemodel(model);
```

```
  /* Free environment */

  GRBfreeenv(env);

  return 0;
}
```

dense_c.c

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* This example formulates and solves the following simple QP model:

     minimize    x + y + x^2 + x*y + y^2 + y*z + z^2
     subject to  x + 2 y + 3 z >= 4
                 x +   y       >= 1

   The example illustrates the use of dense matrices to store A and Q
   (and dense vectors for the other relevant data).  We don't recommend
   that you use dense matrices, but this example may be helpful if you
   already have your data in this format.
*/

#include <stdlib.h>
#include <stdio.h>
#include "gurobi_c.h"

/*
  Solve an LP/QP/MILP/MIQP represented using dense matrices.  This
  routine assumes that A and Q are both stored in row-major order.
  It returns 1 if the optimization succeeds.  When successful,
  it returns the optimal objective value in 'objvalP', and the
  optimal solution vector in 'solution'.
*/

static int
dense_optimize(GRBenv *env,
               int     rows,
               int     cols,
               double *c,     /* linear portion of objective function */
               double *Q,     /* quadratic portion of objective function */
               double *A,     /* constraint matrix */
               char   *sense, /* constraint senses */
               double *rhs,   /* RHS vector */
               double *lb,    /* variable lower bounds */
               double *ub,    /* variable upper bounds */
               char   *vtype, /* variable types (continuous, binary, etc.) */
               double *solution,
               double *objvalP)
{
  GRBmodel *model = NULL;
  int       i, j, optimstatus;
  int       error = 0;
```

```
int       success = 0;

/* Create an empty model */

error = GRBnewmodel(env, &model, "dense", cols, c, lb, ub, vtype, NULL);
if (error) goto QUIT;

error = GRBaddconstrs(model, rows, 0, NULL, NULL, NULL, sense, rhs, NULL);
if (error) goto QUIT;

/* Populate A matrix */

for (i = 0; i < rows; i++) {
  for (j = 0; j < cols; j++) {
    if (A[i*cols+j] != 0) {
      error = GRBchgcoeffs(model, 1, &i, &j, &A[i*cols+j]);
      if (error) goto QUIT;
    }
  }
}

/* Populate Q matrix */

if (Q) {
  for (i = 0; i < cols; i++) {
    for (j = 0; j < cols; j++) {
      if (Q[i*cols+j] != 0) {
        error = GRBaddqpterms(model, 1, &i, &j, &Q[i*cols+j]);
        if (error) goto QUIT;
      }
    }
  }
}

/* Optimize model */

error = GRBoptimize(model);
if (error) goto QUIT;

/* Write model to 'dense.lp' */

error = GRBwrite(model, "dense.lp");
if (error) goto QUIT;

/* Capture solution information */
```

```
  error = GRBgetintattr(model, GRB_INT_ATTR_STATUS, &optimstatus);
  if (error) goto QUIT;

  if (optimstatus == GRB_OPTIMAL) {

    error = GRBgetdblattr(model, GRB_DBL_ATTR_OBJVAL, objvalP);
    if (error) goto QUIT;

    error = GRBgetdblattrarray(model, GRB_DBL_ATTR_X, 0, cols, solution);
    if (error) goto QUIT;

    success = 1;
  }

QUIT:

  /* Error reporting */

  if (error) {
    printf("ERROR: %s\n", GRBgeterrormsg(env));
    exit(1);
  }

  /* Free model */

  GRBfreemodel(model);

  return success;
}

int
main(int   argc,
     char *argv[])
{
  GRBenv *env     = NULL;
  int     error   = 0;
  double  c[]     = {1, 1, 0};
  double  Q[3][3] = {{1, 1, 0}, {0, 1, 1}, {0, 0, 1}};
  double  A[2][3] = {{1, 2, 3}, {1, 1, 0}};
  char    sense[] = {'>', '>'};
  double  rhs[]   = {4, 1};
  double  lb[]    = {0, 0, 0};
  double  sol[3];
  int     solved;
```

```
  double  objval;

  /* Create environment */

  error = GRBloadenv(&env, "dense.log");
  if (error) goto QUIT;

  /* Solve the model */

  solved = dense_optimize(env, 2, 3, c, &Q[0][0], &A[0][0], sense, rhs, lb,
                          NULL, NULL, sol, &objval);

  if (solved)
    printf("Solved: x=%.4f, y=%.4f, z=%.4f\n", sol[0], sol[1], sol[2]);

QUIT:

  /* Free environment */

  GRBfreeenv(env);

  return 0;
}
```

diet_c.c

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* Solve the classic diet model, showing how to add constraints
   to an existing model. */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "gurobi_c.h"

int printSolution(GRBmodel* model, int nCategories, int nFoods);


int
main(int   argc,
     char *argv[])
{
  GRBenv   *env   = NULL;
  GRBmodel *model = NULL;
  int       error = 0;
  int       i, j;
  int      *cbeg, *cind, idx;
  double   *cval, *rhs;
  char     *sense;

  /* Nutrition guidelines, based on
     USDA Dietary Guidelines for Americans, 2005
     http://www.health.gov/DietaryGuidelines/dga2005/ */

  const int nCategories = 4;
  char *Categories[] =
    { "calories", "protein", "fat", "sodium" };
  double minNutrition[] = { 1800, 91, 0, 0 };
  double maxNutrition[] = { 2200, GRB_INFINITY, 65, 1779 };

  /* Set of foods */
  const int nFoods = 9;
  char* Foods[] =
    { "hamburger", "chicken", "hot dog", "fries",
      "macaroni", "pizza", "salad", "milk", "ice cream" };
  double cost[] =
    { 2.49, 2.89, 1.50, 1.89, 2.09, 1.99, 2.49, 0.89, 1.59 };

  /* Nutrition values for the foods */
```

```
double nutritionValues[][4] = {
                                { 410, 24, 26, 730 },
                                { 420, 32, 10, 1190 },
                                { 560, 20, 32, 1800 },
                                { 380, 4, 19, 270 },
                                { 320, 12, 10, 930 },
                                { 320, 15, 12, 820 },
                                { 320, 31, 12, 1230 },
                                { 100, 8, 2.5, 125 },
                                { 330, 8, 10, 180 }
                              };

/* Create environment */
error = GRBloadenv(&env, "diet.log");
if (error) goto QUIT;

/* Create initial model */
error = GRBnewmodel(env, &model, "diet", nFoods + nCategories,
                    NULL, NULL, NULL, NULL, NULL);
if (error) goto QUIT;

/* Initialize decision variables for the foods to buy */
for (j = 0; j < nFoods; ++j)
{
  error = GRBsetdblattrelement(model, "Obj", j, cost[j]);
  if (error) goto QUIT;
  error = GRBsetstrattrelement(model, "VarName", j, Foods[j]);
  if (error) goto QUIT;
}

/* Initialize decision variables for the nutrition information,
   which we limit via bounds */
for (j = 0; j < nCategories; ++j)
{
  error = GRBsetdblattrelement(model, "LB", j + nFoods, minNutrition[j]);
  if (error) goto QUIT;
  error = GRBsetdblattrelement(model, "UB", j + nFoods, maxNutrition[j]);
  if (error) goto QUIT;
  error = GRBsetstrattrelement(model, "VarName", j + nFoods, Categories[j]);
  if (error) goto QUIT;
}

/* The objective is to minimize the costs */
error = GRBsetintattr(model, "ModelSense", GRB_MINIMIZE);
if (error) goto QUIT;
```

```
/* Nutrition constraints */
cbeg = malloc(sizeof(int) * nCategories);
if (!cbeg) goto QUIT;
cind = malloc(sizeof(int) * nCategories * (nFoods + 1));
if (!cind) goto QUIT;
cval = malloc(sizeof(double) * nCategories * (nFoods + 1));
if (!cval) goto QUIT;
rhs = malloc(sizeof(double) * nCategories);
if (!rhs) goto QUIT;
sense = malloc(sizeof(char) * nCategories);
if (!sense) goto QUIT;
idx = 0;
for (i = 0; i < nCategories; ++i)
{
  cbeg[i] = idx;
  rhs[i] = 0.0;
  sense[i] = GRB_EQUAL;
  for (j = 0; j < nFoods; ++j)
  {
    cind[idx] = j;
    cval[idx++] = nutritionValues[j][i];
  }
  cind[idx] = nFoods + i;
  cval[idx++] = -1.0;
}

error = GRBaddconstrs(model, nCategories, idx, cbeg, cind, cval, sense,
                      rhs, Categories);
if (error) goto QUIT;

/* Solve */
error = GRBoptimize(model);
if (error) goto QUIT;
error = printSolution(model, nCategories, nFoods);
if (error) goto QUIT;

printf("\nAdding constraint: at most 6 servings of dairy\n");
cind[0] = 7;
cval[0] = 1.0;
cind[1] = 8;
cval[1] = 1.0;
error = GRBaddconstr(model, 2, cind, cval, GRB_LESS_EQUAL, 6.0,
                     "limit_dairy");
if (error) goto QUIT;
```

```
  /* Solve */
  error = GRBoptimize(model);
  if (error) goto QUIT;
  error = printSolution(model, nCategories, nFoods);
  if (error) goto QUIT;



QUIT:

  /* Error reporting */

  if (error)
  {
    printf("ERROR: %s\n", GRBgeterrormsg(env));
    exit(1);
  }

  /* Free data */

  free(cbeg);
  free(cind);
  free(cval);
  free(rhs);
  free(sense);

  /* Free model */

  GRBfreemodel(model);

  /* Free environment */

  GRBfreeenv(env);

  return 0;
}

int printSolution(GRBmodel* model, int nCategories, int nFoods)
{
  int error, status, i, j;
  double obj, x;
  char* vname;

  error = GRBgetintattr(model, "Status", &status);
```

```
  if (error) return error;
  if (status == GRB_OPTIMAL)
  {
    error = GRBgetdblattr(model, "ObjVal", &obj);
    if (error) return error;
    printf("\nCost: %f\n\nBuy:\n", obj);
    for (j = 0; j < nFoods; ++j)
    {
      error = GRBgetdblattrelement(model, "X", j, &x);
      if (error) return error;
      if (x > 0.0001)
      {
        error = GRBgetstrattrelement(model, "VarName", j, &vname);
        if (error) return error;
        printf("%s %f\n", vname, x);
      }
    }
    printf("\nNutrition:\n");
    for (i = 0; i < nCategories; ++i)
    {
      error = GRBgetdblattrelement(model, "X", i + nFoods, &x);
      if (error) return error;
      error = GRBgetstrattrelement(model, "VarName", i + nFoods, &vname);
      if (error) return error;
      printf("%s %f\n", vname, x);
    }
  }
  else
  {
    printf("No solution\n");
  }

  return 0;
}
```

facility_c.c

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* Facility location: a company currently ships its product from 5 plants
   to 4 warehouses. It is considering closing some plants to reduce
   costs. What plant(s) should the company close, in order to minimize
   transportation and fixed costs?

   Based on an example from Frontline Systems:
   http://www.solver.com/disfacility.htm
   Used with permission.
 */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "gurobi_c.h"


#define opencol(p)         p
#define transportcol(w,p)  nPlants*(w+1)+p
#define MAXSTR             128

int
main(int   argc,
     char *argv[])
{
  GRBenv   *env   = NULL;
  GRBmodel *model = NULL;
  int       error = 0;
  int       p, w, col;
  int      *cbeg = NULL;
  int      *cind = NULL;
  int       idx, rowct;
  double   *cval = NULL;
  double   *rhs = NULL;
  char     *sense = NULL;
  char     vname[MAXSTR];
  int      cnamect = 0;
  char   **cname = NULL;
  double   maxFixed = -GRB_INFINITY, sol, obj;

  /* Number of plants and warehouses */
  const int nPlants = 5;
  const int nWarehouses = 4;
```

```
/* Warehouse demand in thousands of units */
double Demand[] = { 15, 18, 14, 20 };

/* Plant capacity in thousands of units */
double Capacity[] = { 20, 22, 17, 19, 18 };

/* Fixed costs for each plant */
double FixedCosts[] =
  { 12000, 15000, 17000, 13000, 16000 };

/* Transportation costs per thousand units */
double TransCosts[4][5] = {
                            { 4000, 2000, 3000, 2500, 4500 },
                            { 2500, 2600, 3400, 3000, 4000 },
                            { 1200, 1800, 2600, 4100, 3000 },
                            { 2200, 2600, 3100, 3700, 3200 }
                          };

/* Create environment */
error = GRBloadenv(&env, "facility.log");
if (error) goto QUIT;

/* Create initial model */
error = GRBnewmodel(env, &model, "facility", nPlants * (nWarehouses + 1),
                    NULL, NULL, NULL, NULL, NULL);
if (error) goto QUIT;

/* Initialize decision variables for plant open variables */
for (p = 0; p < nPlants; ++p)
{
  col = opencol(p);
  error = GRBsetcharattrelement(model, "VType", col, GRB_BINARY);
  if (error) goto QUIT;
  error = GRBsetdblattrelement(model, "Obj", col, FixedCosts[p]);
  if (error) goto QUIT;
  sprintf(vname, "Open%i", p);
  error = GRBsetstrattrelement(model, "VarName", col, vname);
  if (error) goto QUIT;
}

/* Initialize decision variables for transportation decision variables:
   how much to transport from a plant p to a warehouse w */
for (w = 0; w < nWarehouses; ++w)
{
```

```
  for (p = 0; p < nPlants; ++p)
  {
    col = transportcol(w, p);
    error = GRBsetdblattrelement(model, "Obj", col, TransCosts[w][p]);
    if (error) goto QUIT;
    sprintf(vname, "Trans%i.%i", p, w);
    error = GRBsetstrattrelement(model, "VarName", col, vname);
    if (error) goto QUIT;
  }
}

/* The objective is to minimize the total fixed and variable costs */
error = GRBsetintattr(model, "ModelSense", GRB_MINIMIZE);
if (error) goto QUIT;

/* Make space for constraint data */
rowct = (nPlants > nWarehouses) ? nPlants : nWarehouses;
cbeg = malloc(sizeof(int) * rowct);
if (!cbeg) goto QUIT;
cind = malloc(sizeof(int) * (nPlants * (nWarehouses + 1)));
if (!cind) goto QUIT;
cval = malloc(sizeof(double) * (nPlants * (nWarehouses + 1)));
if (!cval) goto QUIT;
rhs = malloc(sizeof(double) * rowct);
if (!rhs) goto QUIT;
sense = malloc(sizeof(char) * rowct);
if (!sense) goto QUIT;
cname = calloc(rowct, sizeof(char*));
if (!cname) goto QUIT;

/* Production constraints
   Note that the limit sets the production to zero if
   the plant is closed */
idx = 0;
for (p = 0; p < nPlants; ++p)
{
  cbeg[p] = idx;
  rhs[p] = 0.0;
  sense[p] = GRB_LESS_EQUAL;
  cname[p] = malloc(sizeof(char) * MAXSTR);
  if (!cname[p]) goto QUIT;
  cnamect++;
  sprintf(cname[p], "Capacity%i", p);
  for (w = 0; w < nWarehouses; ++w)
  {
```

```
    cind[idx] = transportcol(w, p);
    cval[idx++] = 1.0;
  }
  cind[idx] = opencol(p);
  cval[idx++] = -Capacity[p];
}
error = GRBaddconstrs(model, nPlants, idx, cbeg, cind, cval, sense,
                      rhs, cname);
if (error) goto QUIT;

/* Demand constraints */
idx = 0;
for (w = 0; w < nWarehouses; ++w)
{
  cbeg[w] = idx;
  sense[w] = GRB_EQUAL;
  sprintf(cname[w], "Demand%i", w);
  for (p = 0; p < nPlants; ++p)
  {
    cind[idx] = transportcol(w, p);
    cval[idx++] = 1.0;
  }
}
error = GRBaddconstrs(model, nWarehouses, idx, cbeg, cind, cval, sense,
                      Demand, cname);
if (error) goto QUIT;

/* Guess at the starting point: close the plant with the highest
   fixed costs; open all others */

/* First, open all plants */
for (p = 0; p < nPlants; ++p)
{
  error = GRBsetdblattrelement(model, "Start", opencol(p), 1.0);
  if (error) goto QUIT;
}

/* Now close the plant with the highest fixed cost */
printf("Initial guess:\n");
for (p = 0; p < nPlants; ++p)
{
  if (FixedCosts[p] > maxFixed)
  {
    maxFixed = FixedCosts[p];
  }
```

```
}
for (p = 0; p < nPlants; ++p)
{
  if (FixedCosts[p] == maxFixed)
  {
    error = GRBsetdblattrelement(model, "Start", opencol(p), 0.0);
    if (error) goto QUIT;
    printf("Closing plant %i\n\n", p);
    break;
  }
}

/* Use barrier to solve root relaxation */
error = GRBsetintparam(GRBgetenv(model),
                       GRB_INT_PAR_METHOD,
                       GRB_METHOD_BARRIER);
if (error) goto QUIT;

/* Solve */
error = GRBoptimize(model);
if (error) goto QUIT;

/* Print solution */
error = GRBgetdblattr(model, "ObjVal", &obj);
if (error) goto QUIT;
printf("\nTOTAL COSTS: %f\n", obj);
printf("SOLUTION:\n");
for (p = 0; p < nPlants; ++p)
{
  error = GRBgetdblattrelement(model, "X", opencol(p), &sol);
  if (error) goto QUIT;
  if (sol > 0.99)
  {
    printf("Plant %i open:\n", p);
    for (w = 0; w < nWarehouses; ++w)
    {
      error = GRBgetdblattrelement(model, "X", transportcol(w, p), &sol);
      if (error) goto QUIT;
      if (sol > 0.0001)
      {
        printf("  Transport %f units to warehouse %i\n", sol, w);
      }
    }
  }
  else
```

```
    {
      printf("Plant %i closed!\n", p);
    }
  }


QUIT:

  /* Error reporting */

  if (error)
  {
    printf("ERROR: %s\n", GRBgeterrormsg(env));
    exit(1);
  }

  /* Free data */

  free(cbeg);
  free(cind);
  free(cval);
  free(rhs);
  free(sense);
  for (p = 0; p < cnamect; ++p) {
    free(cname[p]);
  }
  free(cname);

  /* Free model */

  GRBfreemodel(model);

  /* Free environment */

  GRBfreeenv(env);

  return 0;
}
```

feasopt_c.c

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* This example reads a MIP model from a file, adds artificial
   variables to each constraint, and then minimizes the sum of the
   artificial variables.  A solution with objective zero corresponds
   to a feasible solution to the input model.
   We can also use FeasRelax feature to do it. In this example, we
   use minrelax=1, i.e. optimizing the returned model finds a solution
   that minimizes the original objective, but only from among those
   solutions that minimize the sum of the artificial variables. */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "gurobi_c.h"

int
main(int   argc,
     char *argv[])
{
  GRBenv   *env       = NULL;
  GRBmodel *model     = NULL;
  GRBmodel *feasmodel = NULL;
  double   *rhspen      = NULL;
  int       error = 0;
  int       i, j;
  int       numvars, numconstrs;
  char      sense;
  int       vind[1];
  double    vval[1];
  double    feasobj;
  char      *cname, *vname;

  if (argc < 2)
  {
    fprintf(stderr, "Usage: feasopt_c filename\n");
    exit(1);
  }

  error = GRBloadenv(&env, "feasopt.log");
  if (error) goto QUIT;

  error = GRBreadmodel(env, argv[1], &model);
```

```
if (error) goto QUIT;

/* Create a copy to use FeasRelax feature later */

feasmodel = GRBcopymodel(model);
if (error) goto QUIT;

/* clear objective */
error = GRBgetintattr(model, "NumVars", &numvars);
if (error) goto QUIT;
for (j = 0; j < numvars; ++j)
{
  error = GRBsetdblattrelement(model, "Obj", j, 0.0);
  if (error) goto QUIT;
}

/* add slack variables */
error = GRBgetintattr(model, "NumConstrs", &numconstrs);
if (error) goto QUIT;
for (i = 0; i < numconstrs; ++i)
{
  error = GRBgetcharattrelement(model, "Sense", i, &sense);
  if (error) goto QUIT;
  if (sense != '>')
  {
    error = GRBgetstrattrelement(model, "ConstrName", i, &cname);
    if (error) goto QUIT;
    vname = malloc(sizeof(char) * (6 + strlen(cname)));
    if (!vname) goto QUIT;
    strcpy(vname, "ArtN_");
    strcat(vname, cname);
    vind[0] = i;
    vval[0] = -1.0;
    error = GRBaddvar(model, 1, vind, vval, 1.0, 0.0, GRB_INFINITY,
                      GRB_CONTINUOUS, vname);
    if (error) goto QUIT;
    free(vname);
  }
  if (sense != '<')
  {
    error = GRBgetstrattrelement(model, "ConstrName", i, &cname);
    if (error) goto QUIT;
    vname = malloc(sizeof(char) * (6 + strlen(cname)));
    if (!vname) goto QUIT;
    strcpy(vname, "ArtP_");
```

```
      strcat(vname, cname);
      vind[0] = i;
      vval[0] = 1.0;
      error = GRBaddvar(model, 1, vind, vval, 1.0, 0.0, GRB_INFINITY,
                        GRB_CONTINUOUS, vname);
      if (error) goto QUIT;
      free(vname);
    }
  }

  /* Optimize modified model */

  error = GRBoptimize(model);
  if (error) goto QUIT;

  error = GRBwrite(model, "feasopt.lp");
  if (error) goto QUIT;

  /* Use FeasRelax feature */

  rhspen = (double *) malloc(numconstrs*sizeof(double));
  if (rhspen == NULL) {
    printf("ERROR: out of memory\n");
    goto QUIT;
  }

  /* set penalties for artificial variables */
  for (i = 0; i < numconstrs; i++) rhspen[i] = 1;

  /* create a FeasRelax model with the original objective recovered
     and enforcement on minimum of aretificial variables */
  error = GRBfeasrelax(feasmodel, GRB_FEASRELAX_LINEAR, 1,
                       NULL, NULL, rhspen, &feasobj);
  if (error) goto QUIT;

  /* optimize FeasRelax model */
  error = GRBwrite(feasmodel, "feasopt1.lp");
  if (error) goto QUIT;

  error = GRBoptimize(feasmodel);
  if (error) goto QUIT;


QUIT:
```

```
  /* Error reporting */

  if (error)
  {
    printf("ERROR: %s\n", GRBgeterrormsg(env));
    exit(1);
  }

  /* Free models, env and etc. */

  if (rhspen) free(rhspen);

  GRBfreemodel(model);
  GRBfreemodel(feasmodel);

  GRBfreeenv(env);

  return 0;
}
```

fixanddive_c.c

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* Implement a simple MIP heuristic.  Relax the model,
   sort variables based on fractionality, and fix the 25% of
   the fractional variables that are closest to integer variables.
   Repeat until either the relaxation is integer feasible or
   linearly infeasible. */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "gurobi_c.h"

typedef struct
{
  int index;
  double X;
}
var_t ;

int vcomp(const void* v1, const void* v2);


int
main(int   argc,
     char *argv[])
{
  GRBenv   *env   = NULL, *modelenv = NULL;
  GRBmodel *model = NULL;
  int       error = 0;
  int       j, iter, nfix;
  int       numvars, numintvars, numfractional;
  int      *intvars = NULL;
  int       status;
  char      vtype, *vname;
  double    sol, obj, fixval;
  var_t    *fractional = NULL;

  if (argc < 2)
  {
    fprintf(stderr, "Usage: fixanddive_c filename\n");
    exit(1);
  }
```

```
error = GRBloadenv(&env, "fixanddive.log");
if (error) goto QUIT;

/* Read model */
error = GRBreadmodel(env, argv[1], &model);
if (error) goto QUIT;

/* Collect integer variables and relax them */
error = GRBgetintattr(model, "NumVars", &numvars);
if (error) goto QUIT;
error = GRBgetintattr(model, "NumIntVars", &numintvars);
if (error) goto QUIT;
intvars = malloc(sizeof(int) * numintvars);
if (!intvars) goto QUIT;
fractional = malloc(sizeof(var_t) * numintvars);
if (!fractional) goto QUIT;
numfractional = 0;
for (j = 0; j < numvars; j++)
{
  error = GRBgetcharattrelement(model, "VType", j, &vtype);
  if (error) goto QUIT;
  if (vtype != GRB_CONTINUOUS)
  {
    intvars[numfractional++] = j;
    error = GRBsetcharattrelement(model, "VType", j, GRB_CONTINUOUS);
    if (error) goto QUIT;
  }
}

modelenv = GRBgetenv(model);
if (!modelenv) goto QUIT;
error = GRBsetintparam(modelenv, "OutputFlag", 0);
if (error) goto QUIT;
error = GRBoptimize(model);
if (error) goto QUIT;

/* Perform multiple iterations. In each iteration, identify the first
   quartile of integer variables that are closest to an integer value
   in the relaxation, fix them to the nearest integer, and repeat. */

for (iter = 0; iter < 1000; ++iter)
{

  /* create a list of fractional variables, sorted in order of
     increasing distance from the relaxation solution to the nearest
```

```
   integer value */

numfractional = 0;
for (j = 0; j < numintvars; ++j)
{
  error = GRBgetdblattrelement(model, "X", intvars[j], &sol);
  if (error) goto QUIT;
  if (fabs(sol - floor(sol + 0.5)) > 1e-5)
  {
    fractional[numfractional].index = intvars[j];
    fractional[numfractional++].X = sol;
  }
}

error = GRBgetdblattr(model, "ObjVal", &obj);
if (error) goto QUIT;
printf("Iteration %i, obj %f, fractional %i\n",
       iter, obj, numfractional);

if (numfractional == 0)
{
  printf("Found feasible solution - objective %f\n", obj);
  break;
}

/* Fix the first quartile to the nearest integer value */
qsort(fractional, numfractional, sizeof(var_t), vcomp);
nfix = numfractional / 4;
nfix = (nfix > 1) ? nfix : 1;
for (j = 0; j < nfix; ++j)
{
  fixval = floor(fractional[j].X + 0.5);
  error = GRBsetdblattrelement(model, "LB", fractional[j].index, fixval);
  if (error) goto QUIT;
  error = GRBsetdblattrelement(model, "UB", fractional[j].index, fixval);
  if (error) goto QUIT;
  error = GRBgetstrattrelement(model, "VarName",
                               fractional[j].index, &vname);
  if (error) goto QUIT;
  printf("  Fix %s to %f ( rel %f )\n", vname, fixval, fractional[j].X);
}

error = GRBoptimize(model);
if (error) goto QUIT;
```

```
    /* Check optimization result */

    error = GRBgetintattr(model, "Status", &status);
    if (error) goto QUIT;
    if (status != GRB_OPTIMAL)
    {
      printf("Relaxation is infeasible\n");
      break;
    }
  }


QUIT:

  /* Error reporting */

  if (error)
  {
    printf("ERROR: %s\n", GRBgeterrormsg(env));
    exit(1);
  }

  /* Free data */

  free(intvars);
  free(fractional);

  /* Free model */

  GRBfreemodel(model);

  /* Free environment */

  GRBfreeenv(env);

  return 0;
}


int vcomp(const void* v1, const void* v2)
{
  double sol1, sol2, frac1, frac2;
  sol1 = fabs(((var_t *)v1)->X);
  sol2 = fabs(((var_t *)v2)->X);
  frac1 = fabs(sol1 - floor(sol1 + 0.5));
```

```
  frac2 = fabs(sol2 - floor(sol2 + 0.5));
  return (frac1 < frac2) ? -1 : ((frac1 == frac2) ? 0 : 1);
}
```

genconstr_c.c

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* In this example we show the use of general constraints for modeling
 * some common expressions. We use as an example a SAT-problem where we
 * want to see if it is possible to satisfy at least four (or all) clauses
 * of the logical for
 *
 * L = (x0 or ~x1 or x2)  and (x1 or ~x2 or x3)  and
 *     (x2 or ~x3 or x0)  and (x3 or ~x0 or x1)  and
 *     (~x0 or ~x1 or x2) and (~x1 or ~x2 or x3) and
 *     (~x2 or ~x3 or x0) and (~x3 or ~x0 or x1)
 *
 * We do this by introducing two variables for each literal (itself and its
 * negated value), a variable for each clause, and then two
 * variables for indicating if we can satisfy four, and another to identify
 * the minimum of the clauses (so if it one, we can satisfy all clauses)
 * and put these two variables in the objective.
 * i.e. the Objective function will be
 *
 * maximize Obj0 + Obj1
 *
 *  Obj0 = MIN(Clause1, ... , Clause8)
 *  Obj1 = 1 -> Clause1 + ... + Clause8 >= 4
 *
 * thus, the objective value will be two if and only if we can satisfy all
 * clauses; one if and only if at least four clauses can be satisfied, and
 * zero otherwise.
 */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "gurobi_c.h"

#define MAXSTR    128
#define NLITERALS 4
#define NCLAUSES  8
#define NOBJ      2
#define NVARS     (2 * NLITERALS + NCLAUSES + NOBJ)
#define LIT(n)    (n)
#define NOTLIT(n) (NLITERALS + n)
#define CLA(n)    (2 * NLITERALS + n)
#define OBJ(n)    (2 * NLITERALS + NCLAUSES + n)
```

```
int
main(void)
{
  GRBenv   *env   = NULL;
  GRBmodel *model = NULL;
  int       error = 0;
  int       cind[NVARS];
  double    cval[NVARS];
  char      buffer[MAXSTR];
  int col, i, status, nSolutions;
  double objval;

  /* Example data */
  const int Clauses[][3] = {{LIT(0), NOTLIT(1), LIT(2)},
                            {LIT(1), NOTLIT(2), LIT(3)},
                            {LIT(2), NOTLIT(3), LIT(0)},
                            {LIT(3), NOTLIT(0), LIT(1)},
                            {NOTLIT(0), NOTLIT(1), LIT(2)},
                            {NOTLIT(1), NOTLIT(2), LIT(3)},
                            {NOTLIT(2), NOTLIT(3), LIT(0)},
                            {NOTLIT(3), NOTLIT(0), LIT(1)}};

  /* Create environment */
  error = GRBloadenv(&env, "genconstr_c.log");
  if (error) goto QUIT;

  /* Create initial model */
  error = GRBnewmodel(env, &model, "genconstr_c", NVARS, NULL,
                      NULL, NULL, NULL, NULL);
  if (error) goto QUIT;

  /* Initialize decision variables and objective */
  for (i = 0; i < NLITERALS; i++) {
    col = LIT(i);
    sprintf(buffer, "X%d", i);
    error = GRBsetcharattrelement(model, "VType", col, GRB_BINARY);
    if (error) goto QUIT;

    error = GRBsetstrattrelement(model, "VarName", col, buffer);
    if (error) goto QUIT;

    col = NOTLIT(i);
    sprintf(buffer, "notX%d", i);
```

```
  error = GRBsetcharattrelement(model, "VType", col, GRB_BINARY);
  if (error) goto QUIT;

  error = GRBsetstrattrelement(model, "VarName", col, buffer);
  if (error) goto QUIT;
}

for (i = 0; i < NCLAUSES; i++) {
  col = CLA(i);
  sprintf(buffer, "Clause%d", i);
  error = GRBsetcharattrelement(model, "VType", col, GRB_BINARY);
  if (error) goto QUIT;

  error = GRBsetstrattrelement(model, "VarName", col, buffer);
  if (error) goto QUIT;
}

for (i = 0; i < NOBJ; i++) {
  col = OBJ(i);
  sprintf(buffer, "Obj%d", i);
  error = GRBsetcharattrelement(model, "VType", col, GRB_BINARY);
  if (error) goto QUIT;

  error = GRBsetstrattrelement(model, "VarName", col, buffer);
  if (error) goto QUIT;

  error = GRBsetdblattrelement(model, "Obj", col, 1.0);
  if (error) goto QUIT;
}

/* Link Xi and notXi */
for (i = 0; i < NLITERALS; i++) {
  sprintf(buffer,"CNSTR_X%d",i);
  cind[0] = LIT(i);
  cind[1] = NOTLIT(i);
  cval[0] = cval[1] = 1;
  error = GRBaddconstr(model, 2, cind, cval, GRB_EQUAL, 1.0, buffer);
  if (error) goto QUIT;
}

/* Link clauses and literals */
for (i = 0; i < NCLAUSES; i++) {
  sprintf(buffer,"CNSTR_Clause%d",i);
  error = GRBaddgenconstrOr(model, buffer, CLA(i), 3, Clauses[i]);
  if (error) goto QUIT;
```

```
}

/* Link objs with clauses */
for (i = 0; i < NCLAUSES; i++) {
  cind[i] = CLA(i);
  cval[i] = 1;
}
error = GRBaddgenconstrMin(model, "CNSTR_Obj0", OBJ(0), NCLAUSES, cind, GRB_INFINITY);
if (error) goto QUIT;

/* note that passing 4 instead of 4.0 will produce undefined behavior */
error = GRBaddgenconstrIndicator(model, "CNSTR_Obj1",
                                 OBJ(1), 1, NCLAUSES, cind, cval,
                                 GRB_GREATER_EQUAL, 4.0);
if (error) goto QUIT;

/* Set global objective sense */
error = GRBsetintattr(model, GRB_INT_ATTR_MODELSENSE, GRB_MAXIMIZE);
if (error) goto QUIT;

/* Save problem */
error = GRBwrite(model, "genconstr_c.mps");
if (error) goto QUIT;

error = GRBwrite(model, "genconstr_c.lp");
if (error) goto QUIT;

/* Optimize */
error = GRBoptimize(model);
if (error) goto QUIT;

/* Status checking */
error = GRBgetintattr(model, "Status", &status);
if (error) goto QUIT;

if (status == GRB_INF_OR_UNBD ||
    status == GRB_INFEASIBLE  ||
    status == GRB_UNBOUNDED     ) {
  printf("The model cannot be solved "
         "because it is infeasible or unbounded\n");
  goto QUIT;
}
if (status != GRB_OPTIMAL) {
  printf("Optimization was stopped with status %i\n", status);
  goto QUIT;
```

```
  }

  /* Print result */
  error = GRBgetdblattr(model, GRB_DBL_ATTR_OBJVAL, &objval);
  if (error) goto QUIT;

  if (objval > 1.9)
    printf("Logical expression is satisfiable\n");
  else if (objval > 0.9)
    printf("At least four clauses can be satisfied\n");
  else
    printf("Not even three clauses can be satisfied\n");

QUIT:

  if (model != NULL) GRBfreemodel(model);
  if (env != NULL)   GRBfreeenv(env);

  return error;
}
```

lp_c.c

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* This example reads an LP model from a file and solves it.
   If the model is infeasible or unbounded, the example turns off
   presolve and solves the model again. If the model is infeasible,
   the example computes an Irreducible Inconsistent Subsystem (IIS),
   and writes it to a file */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "gurobi_c.h"

int
main(int   argc,
     char *argv[])
{
  GRBenv   *masterenv = NULL;
  GRBmodel *model     = NULL;
  GRBenv   *modelenv  = NULL;
  int       error     = 0;
  int       optimstatus;
  double    objval;

  if (argc < 2) {
    fprintf(stderr, "Usage: lp_c filename\n");
    exit(1);
  }

  /* Create environment */

  error = GRBloadenv(&masterenv, "lp.log");
  if (error) goto QUIT;

  /* Read model from file */

  error = GRBreadmodel(masterenv, argv[1], &model);
  if (error) goto QUIT;

  /* Solve model */

  error = GRBoptimize(model);
  if (error) goto QUIT;
```

```
/* Capture solution information */

error = GRBgetintattr(model, GRB_INT_ATTR_STATUS, &optimstatus);
if (error) goto QUIT;

/* If model is infeasible or unbounded, turn off presolve and resolve */

if (optimstatus == GRB_INF_OR_UNBD) {
  modelenv = GRBgetenv(model);
  if (!modelenv) {
    fprintf(stderr, "Error: could not get model environment\n");
    goto QUIT;
  }

  /* Change parameter on model environment.  The model now has
     a copy of the master environment, so changing the master will
     no longer affect the model.  */

  error = GRBsetintparam(modelenv, "PRESOLVE", 0);
  if (error) goto QUIT;

  error = GRBoptimize(model);
  if (error) goto QUIT;

  error = GRBgetintattr(model, GRB_INT_ATTR_STATUS, &optimstatus);
  if (error) goto QUIT;
}

if (optimstatus == GRB_OPTIMAL) {
  error = GRBgetdblattr(model, GRB_DBL_ATTR_OBJVAL, &objval);
  if (error) goto QUIT;
  printf("Optimal objective: %.4e\n\n", objval);
} else if (optimstatus == GRB_INFEASIBLE) {
  printf("Model is infeasible\n\n");

  error = GRBcomputeIIS(model);
  if (error) goto QUIT;

  error = GRBwrite(model, "model.ilp");
  if (error) goto QUIT;
} else if (optimstatus == GRB_UNBOUNDED) {
  printf("Model is unbounded\n\n");
} else {
  printf("Optimization was stopped with status = %d\n\n", optimstatus);
}
```

```
QUIT:

  /* Error reporting */

  if (error) {
    printf("ERROR: %s\n", GRBgeterrormsg(masterenv));
    exit(1);
  }

  /* Free model */

  GRBfreemodel(model);

  /* Free environment */

  GRBfreeenv(masterenv);

  return 0;
}
```

lpmethod_c.c

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* Solve a model with different values of the Method parameter;
   show which value gives the shortest solve time. */

#include <stdlib.h>
#include <stdio.h>
#include "gurobi_c.h"

int
main(int   argc,
     char *argv[])
{
  GRBenv   *env = NULL, *menv;
  GRBmodel *m = NULL;
  int       error = 0;
  int       i;
  int       optimstatus;
  int       bestMethod = -1;
  double    bestTime;

  if (argc < 2)
  {
    fprintf(stderr, "Usage: lpmethod_c filename\n");
    exit(1);
  }

  error = GRBloadenv(&env, "lpmethod.log");
  if (error) goto QUIT;

  /* Read model */
  error = GRBreadmodel(env, argv[1], &m);
  if (error) goto QUIT;
  menv = GRBgetenv(m);
  error = GRBgetdblparam(menv, "TimeLimit", &bestTime);
  if (error) goto QUIT;

  /* Solve the model with different values of Method */
  for (i = 0; i <= 2; ++i)
  {
    error = GRBresetmodel(m);
    if (error) goto QUIT;
    error = GRBsetintparam(menv, "Method", i);
    if (error) goto QUIT;
```

```
    error = GRBoptimize(m);
    if (error) goto QUIT;
    error = GRBgetintattr(m, "Status", &optimstatus);
    if (error) goto QUIT;
    if (optimstatus == GRB_OPTIMAL) {
      error = GRBgetdblattr(m, "Runtime", &bestTime);
      if (error) goto QUIT;
      bestMethod = i;
      /* Reduce the TimeLimit parameter to save time
         with other methods */
      error = GRBsetdblparam(menv, "TimeLimit", bestTime);
      if (error) goto QUIT;
    }
  }

  /* Report which method was fastest */
  if (bestMethod == -1) {
    printf("Unable to solve this model\n");
  } else {
    printf("Solved in %f seconds with Method: %i\n",
           bestTime, bestMethod);
  }

QUIT:

  /* Error reporting */

  if (error)
  {
    printf("ERROR: %s\n", GRBgeterrormsg(env));
    exit(1);
  }

  /* Free model */

  GRBfreemodel(m);

  /* Free environment */

  GRBfreeenv(env);

  return 0;
}
```

lpmod_c.c

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* This example reads an LP model from a file and solves it.
   If the model can be solved, then it finds the smallest positive variable,
   sets its upper bound to zero, and resolves the model two ways:
   first with an advanced start, then without an advanced start
   (i.e. 'from scratch'). */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "gurobi_c.h"

int
main(int   argc,
     char *argv[])
{
  GRBenv   *env   = NULL;
  GRBmodel *model = NULL;
  int       error = 0;
  int       j;
  int       numvars, isMIP, status, minVar = 0;
  double    minVal = GRB_INFINITY, sol, lb;
  char     *varname;
  double    warmCount, warmTime, coldCount, coldTime;

  if (argc < 2)
  {
    fprintf(stderr, "Usage: lpmod_c filename\n");
    exit(1);
  }

  error = GRBloadenv(&env, "lpmod.log");
  if (error) goto QUIT;

  /* Read model and determine whether it is an LP */
  error = GRBreadmodel(env, argv[1], &model);
  if (error) goto QUIT;
  error = GRBgetintattr(model, "IsMIP", &isMIP);
  if (error) goto QUIT;
  if (isMIP)
  {
    printf("The model is not a linear program\n");
```

```
  goto QUIT;
}

error = GRBoptimize(model);
if (error) goto QUIT;

error = GRBgetintattr(model, "Status", &status);
if (error) goto QUIT;

if ((status == GRB_INF_OR_UNBD) || (status == GRB_INFEASIBLE) ||
    (status == GRB_UNBOUNDED))
{
  printf("The model cannot be solved because it is ");
  printf("infeasible or unbounded\n");
  goto QUIT;
}

if (status != GRB_OPTIMAL)
{
  printf("Optimization was stopped with status %i\n", status);
  goto QUIT;
}

/* Find the smallest variable value */
error = GRBgetintattr(model, "NumVars", &numvars);
if (error) goto QUIT;
for (j = 0; j < numvars; ++j)
{
  error = GRBgetdblattrelement(model, "X", j, &sol);
  if (error) goto QUIT;
  error = GRBgetdblattrelement(model, "LB", j, &lb);
  if (error) goto QUIT;
  if ((sol > 0.0001) && (sol < minVal) &&
      (lb == 0.0))
  {
    minVal = sol;
    minVar = j;
  }
}

error = GRBgetstrattrelement(model, "VarName", minVar, &varname);
if (error) goto QUIT;
printf("\n*** Setting %s from %f to zero ***\n\n", varname, minVal);
error = GRBsetdblattrelement(model, "LB", minVar, 0.0);
if (error) goto QUIT;
```

```
  /* Solve from this starting point */
  error = GRBoptimize(model);
  if (error) goto QUIT;

  /* Save iteration & time info */
  error = GRBgetdblattr(model, "IterCount", &warmCount);
  if (error) goto QUIT;
  error = GRBgetdblattr(model, "Runtime", &warmTime);
  if (error) goto QUIT;

  /* Reset the model and resolve */
  printf("\n*** Resetting and solving ");
  printf("without an advanced start ***\n\n");
  error = GRBresetmodel(model);
  if (error) goto QUIT;
  error = GRBoptimize(model);
  if (error) goto QUIT;

  /* Save iteration & time info */
  error = GRBgetdblattr(model, "IterCount", &coldCount);
  if (error) goto QUIT;
  error = GRBgetdblattr(model, "Runtime", &coldTime);
  if (error) goto QUIT;

  printf("\n*** Warm start: %f iterations, %f seconds\n",
         warmCount, warmTime);
  printf("*** Cold start: %f iterations, %f seconds\n",
         coldCount, coldTime);


QUIT:

  /* Error reporting */

  if (error)
  {
    printf("ERROR: %s\n", GRBgeterrormsg(env));
    exit(1);
  }

  /* Free model */

  GRBfreemodel(model);
```

```
  /* Free environment */

  GRBfreeenv(env);

  return 0;
}
```

mip1_c.c

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* This example formulates and solves the following simple MIP model:

     maximize    x +   y + 2 z
     subject to  x + 2 y + 3 z <= 4
                 x +   y       >= 1
     x, y, z binary
*/

#include <stdlib.h>
#include <stdio.h>
#include "gurobi_c.h"

int
main(int   argc,
     char *argv[])
{
  GRBenv   *env   = NULL;
  GRBmodel *model = NULL;
  int       error = 0;
  double    sol[3];
  int       ind[3];
  double    val[3];
  double    obj[3];
  char      vtype[3];
  int       optimstatus;
  double    objval;

  /* Create environment */

  error = GRBloadenv(&env, "mip1.log");
  if (error) goto QUIT;

  /* Create an empty model */

  error = GRBnewmodel(env, &model, "mip1", 0, NULL, NULL, NULL, NULL, NULL);
  if (error) goto QUIT;


  /* Add variables */

  obj[0] = 1; obj[1] = 1; obj[2] = 2;
  vtype[0] = GRB_BINARY; vtype[1] = GRB_BINARY; vtype[2] = GRB_BINARY;
```

```
error = GRBaddvars(model, 3, 0, NULL, NULL, NULL, obj, NULL, NULL, vtype,
                   NULL);
if (error) goto QUIT;

/* Change objective sense to maximization */

error = GRBsetintattr(model, GRB_INT_ATTR_MODELSENSE, GRB_MAXIMIZE);
if (error) goto QUIT;

/* First constraint: x + 2 y + 3 z <= 4 */

ind[0] = 0; ind[1] = 1; ind[2] = 2;
val[0] = 1; val[1] = 2; val[2] = 3;

error = GRBaddconstr(model, 3, ind, val, GRB_LESS_EQUAL, 4.0, "c0");
if (error) goto QUIT;

/* Second constraint: x + y >= 1 */

ind[0] = 0; ind[1] = 1;
val[0] = 1; val[1] = 1;

error = GRBaddconstr(model, 2, ind, val, GRB_GREATER_EQUAL, 1.0, "c1");
if (error) goto QUIT;

/* Optimize model */

error = GRBoptimize(model);
if (error) goto QUIT;

/* Write model to 'mip1.lp' */

error = GRBwrite(model, "mip1.lp");
if (error) goto QUIT;

/* Capture solution information */

error = GRBgetintattr(model, GRB_INT_ATTR_STATUS, &optimstatus);
if (error) goto QUIT;

error = GRBgetdblattr(model, GRB_DBL_ATTR_OBJVAL, &objval);
if (error) goto QUIT;

error = GRBgetdblattrarray(model, GRB_DBL_ATTR_X, 0, 3, sol);
if (error) goto QUIT;
```

```
  printf("\nOptimization complete\n");
  if (optimstatus == GRB_OPTIMAL) {
    printf("Optimal objective: %.4e\n", objval);

    printf("  x=%.0f, y=%.0f, z=%.0f\n", sol[0], sol[1], sol[2]);
  } else if (optimstatus == GRB_INF_OR_UNBD) {
    printf("Model is infeasible or unbounded\n");
  } else {
    printf("Optimization was stopped early\n");
  }

QUIT:

  /* Error reporting */

  if (error) {
    printf("ERROR: %s\n", GRBgeterrormsg(env));
    exit(1);
  }

  /* Free model */

  GRBfreemodel(model);

  /* Free environment */

  GRBfreeenv(env);

  return 0;
}
```

mip2_c.c

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* This example reads a MIP model from a file, solves it and
   prints the objective values from all feasible solutions
   generated while solving the MIP. Then it creates the fixed
   model and solves that model. */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "gurobi_c.h"

int
main(int   argc,
     char *argv[])
{
  GRBenv   *env   = NULL;
  GRBmodel *model = NULL;
  GRBmodel *fixed = NULL;
  int       error = 0;
  int       ismip;
  int       j, k, solcount, numvars;
  double    objn, vobj, xn;
  int       optimstatus, foptimstatus;
  double    objval, fobjval;
  char      *varname;
  double    x;

  /* To change settings for a loaded model, we need to get
     the model environment, which will be freed when the model
     is freed. */

  GRBenv   *menv, *fenv;

  if (argc < 2) {
    fprintf(stderr, "Usage: mip2_c filename\n");
    exit(1);
  }

  /* Create environment */

  error = GRBloadenv(&env, "mip2.log");
  if (error) goto QUIT;
```

```
/* Read model from file */

error = GRBreadmodel(env, argv[1], &model);
if (error) goto QUIT;

error = GRBgetintattr(model, "IsMIP", &ismip);
if (error) goto QUIT;

if (ismip == 0) {
  printf("Model is not a MIP\n");
  goto QUIT;
}

/* Get model environment */

menv = GRBgetenv(model);
if (!menv) {
  fprintf(stderr, "Error: could not get model environment\n");
  goto QUIT;
}

/* Solve model */

error = GRBoptimize(model);
if (error) goto QUIT;

/* Capture solution information */

error = GRBgetintattr(model, GRB_INT_ATTR_STATUS, &optimstatus);
if (error) goto QUIT;

printf("\nOptimization complete\n");
if (optimstatus == GRB_OPTIMAL) {
  error = GRBgetdblattr(model, GRB_DBL_ATTR_OBJVAL, &objval);
  if (error) goto QUIT;
  printf("Optimal objective: %.4e\n\n", objval);
} else if (optimstatus == GRB_INF_OR_UNBD) {
  printf("Model is infeasible or unbounded\n\n");
  goto QUIT;
} else if (optimstatus == GRB_INFEASIBLE) {
  printf("Model is infeasible\n\n");
  goto QUIT;
} else if (optimstatus == GRB_UNBOUNDED) {
  printf("Model is unbounded\n\n");
  goto QUIT;
```

```
} else {
  printf("Optimization was stopped with status = %d\n\n", optimstatus);
  goto QUIT;
}

/* Iterate over the solutions and compute the objectives */

error = GRBsetintparam(menv, "OutputFlag", 0);
if (error) goto QUIT;
error = GRBgetintattr(model, "SolCount", &solcount);
if (error) goto QUIT;
error = GRBgetintattr(model, "NumVars", &numvars);
if (error) goto QUIT;

printf("\n");
for ( k = 0; k < solcount; ++k ) {
  error = GRBsetintparam(menv, "SolutionNumber", k);
  objn = 0.0;
  for ( j = 0; j < numvars; ++j ) {
    error = GRBgetdblattrelement(model, "Obj", j, &vobj);
    if (error) goto QUIT;
    error = GRBgetdblattrelement(model, "Xn", j, &xn);
    if (error) goto QUIT;
    objn += vobj * xn;
  }
  printf("Solution %i has objective: %f\n", k, objn);
}
printf("\n");

error = GRBsetintparam(menv, "OutputFlag", 1);
if (error) goto QUIT;

/* Create a fixed model, turn off presolve and solve */

fixed = GRBfixedmodel(model);
if (!fixed) {
  fprintf(stderr, "Error: could not create fixed model\n");
  goto QUIT;
}

fenv = GRBgetenv(fixed);
if (!fenv) {
  fprintf(stderr, "Error: could not get fixed model environment\n");
  goto QUIT;
}
```

```
  error = GRBsetintparam(fenv, "PRESOLVE", 0);
  if (error) goto QUIT;

  error = GRBoptimize(fixed);
  if (error) goto QUIT;

  error = GRBgetintattr(fixed, GRB_INT_ATTR_STATUS, &foptimstatus);
  if (error) goto QUIT;

  if (foptimstatus != GRB_OPTIMAL) {
    fprintf(stderr, "Error: fixed model isn't optimal\n");
    goto QUIT;
  }

  error = GRBgetdblattr(fixed, GRB_DBL_ATTR_OBJVAL, &fobjval);
  if (error) goto QUIT;

  if (fabs(fobjval - objval) > 1.0e-6 * (1.0 + fabs(objval))) {
    fprintf(stderr, "Error: objective values are different\n");
  }

  /* Print values of nonzero variables */
  for ( j = 0; j < numvars; ++j ) {
    error = GRBgetstrattrelement(fixed, "VarName", j, &varname);
    if (error) goto QUIT;
    error = GRBgetdblattrelement(fixed, "X", j, &x);
    if (error) goto QUIT;
    if (x != 0.0) {
      printf("%s %f\n", varname, x);
    }
  }


QUIT:

  /* Error reporting */

  if (error) {
    printf("ERROR: %s\n", GRBgeterrormsg(env));
    exit(1);
  }

  /* Free models */
```

```
  GRBfreemodel(model);
  GRBfreemodel(fixed);

  /* Free environment */

  GRBfreeenv(env);

  return 0;
}
```

multiobj_c.c

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* Want to cover three different sets but subject to a common budget of
 * elements allowed to be used. However, the sets have different priorities to
 * be covered; and we tackle this by using multi-objective optimization. */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "gurobi_c.h"

#define MAXSTR 128

int
main(void)
{
  GRBenv   *env   = NULL;
  GRBenv   *menv  = NULL;
  GRBmodel *model = NULL;
  int       error = 0;
  int      *cbeg  = NULL;
  int      *cind  = NULL;
  double   *cval  = NULL;
  char      buffer[MAXSTR];
  int e, i, status, nSolutions;
  double objn;

  /* Sample data */
  const int groundSetSize = 20;
  const int nSubsets      = 4;
  const int Budget        = 12;
  double Set[][20] =
    { { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
      { 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1 },
      { 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0 },
      { 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0 } };
  int    SetObjPriority[] = {3, 2, 2, 1};
  double SetObjWeight[]   = {1.0, 0.25, 1.25, 1.0};

  /* Create environment */
  error = GRBloadenv(&env, "multiobj_c.log");
  if (error) goto QUIT;
```

```
/* Create initial model */
error = GRBnewmodel(env, &model, "multiobj_c", groundSetSize, NULL,
                    NULL, NULL, NULL, NULL);
if (error) goto QUIT;

/* get model environment */
menv = GRBgetenv(model);
if (!menv) {
  fprintf(stderr, "Error: could not get model environment\n");
  goto QUIT;
}

/* Initialize decision variables for ground set:
 * x[e] == 1 if element e is chosen for the covering. */
for (e = 0; e < groundSetSize; e++) {
  sprintf(buffer, "El%d", e);
  error = GRBsetcharattrelement(model, "VType", e, GRB_BINARY);
  if (error) goto QUIT;

  error = GRBsetstrattrelement(model, "VarName", e, buffer);
  if (error) goto QUIT;
}

/* Make space for constraint data */
cind = malloc(sizeof(int) * groundSetSize);
if (!cind) goto QUIT;
cval = malloc(sizeof(double) * groundSetSize);
if (!cval) goto QUIT;

/* Constraint: limit total number of elements to be picked to be at most
 * Budget */
for (e = 0; e < groundSetSize; e++) {
  cind[e] = e;
  cval[e] = 1.0;
}
sprintf (buffer, "Budget");
error = GRBaddconstr(model, groundSetSize, cind, cval, GRB_LESS_EQUAL,
                     (double)Budget, buffer);
if (error) goto QUIT;

/* Set global sense for ALL objectives */
error = GRBsetintattr(model, GRB_INT_ATTR_MODELSENSE, GRB_MAXIMIZE);
if (error) goto QUIT;

/* Limit how many solutions to collect */
```

```
error = GRBsetintparam(menv, GRB_INT_PAR_POOLSOLUTIONS, 100);
if (error) goto QUIT;

/* Set and configure i-th objective */
for (i = 0; i < nSubsets; i++) {
  sprintf(buffer, "Set%d", i+1);

  error = GRBsetobjectiven(model, i, SetObjPriority[i], SetObjWeight[i],
                           1.0 + i, 0.01, buffer, 0.0, groundSetSize,
                           cind, Set[i]);
  if (error) goto QUIT;
}

/* Save problem */
error = GRBwrite(model, "multiobj_c.lp");
if (error) goto QUIT;
error = GRBwrite(model, "multiobj_c.mps");
if (error) goto QUIT;

/* Optimize */
error = GRBoptimize(model);
if (error) goto QUIT;

/* Status checking */
error = GRBgetintattr(model, "Status", &status);
if (error) goto QUIT;

if (status == GRB_INF_OR_UNBD ||
    status == GRB_INFEASIBLE  ||
    status == GRB_UNBOUNDED     ) {
  printf("The model cannot be solved "
         "because it is infeasible or unbounded\n");
  goto QUIT;
}
if (status != GRB_OPTIMAL) {
  printf("Optimization was stopped with status %i\n", status);
  goto QUIT;
}

/* Print best selected set */
error = GRBgetdblattrarray(model, GRB_DBL_ATTR_X, 0, groundSetSize, cval);
if (error) goto QUIT;

printf("Selected elements in best solution:\n\t");
for (e = 0; e < groundSetSize; e++) {
```

```
    if (cval[e] < .9) continue;
    printf("El%d ", e);
  }

  /* Print number of solutions stored */
  error = GRBgetintattr(model, GRB_INT_ATTR_SOLCOUNT, &nSolutions);
  if (error) goto QUIT;
  printf("\nNumber of solutions found: %d\n", nSolutions);

  /* Print objective values of solutions */

  if (nSolutions > 10) nSolutions = 10;
  printf("Objective values for first %d solutions):\n", nSolutions);
  for (i = 0; i < nSubsets; i++) {
    error = GRBsetintparam(menv, GRB_INT_PAR_OBJNUMBER, i);
    if (error) goto QUIT;

    printf("\tSet %d:", i);
    for (e = 0; e < nSolutions; e++) {
      error = GRBsetintparam(menv, GRB_INT_PAR_SOLUTIONNUMBER, e);
      if (error) goto QUIT;

      error = GRBgetdblattr(model, GRB_DBL_ATTR_OBJNVAL, &objn);
      if (error) goto QUIT;

      printf(" %6g", objn);
    }
    printf("\n");
  }

QUIT:

  if (cind != NULL)  free(cind);
  if (cval != NULL)  free(cval);
  if (model != NULL) GRBfreemodel(model);
  if (env != NULL)   GRBfreeenv(env);

  return error;
}
```

params_c.c

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* Use parameters that are associated with a model.

   A MIP is solved for a few seconds with different sets of parameters.
   The one with the smallest MIP gap is selected, and the optimization
   is resumed until the optimal solution is found.
*/

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "gurobi_c.h"

int
main(int   argc,
     char *argv[])
{
  GRBenv   *env   = NULL, *modelenv = NULL, *bestenv = NULL;
  GRBmodel *model = NULL, *bestmodel = NULL;
  int       error = 0;
  int       ismip, i, mipfocus;
  double    bestgap, gap;

  if (argc < 2)
  {
    fprintf(stderr, "Usage: params_c filename\n");
    exit(1);
  }

  error = GRBloadenv(&env, "params.log");
  if (error) goto QUIT;

  /* Read model and verify that it is a MIP */
  error = GRBreadmodel(env, argv[1], &model);
  if (error) goto QUIT;
  error = GRBgetintattr(model, "IsMIP", &ismip);
  if (error) goto QUIT;
  if (ismip == 0)
  {
    printf("The model is not an integer program\n");
    exit(1);
  }
```

```
/* Set a 2 second time limit */
modelenv = GRBgetenv(model);
if (!modelenv) {
  printf("Cannot retrieve model environment\n");
  exit(1);
}
error = GRBsetdblparam(modelenv, "TimeLimit", 2);
if (error) goto QUIT;

/* Now solve the model with different values of MIPFocus */
bestmodel = GRBcopymodel(model);
if (!bestmodel) {
  printf("Cannot copy model\n");
  exit(1);
}
error = GRBoptimize(bestmodel);
if (error) goto QUIT;
error = GRBgetdblattr(bestmodel, "MIPGap", &bestgap);
if (error) goto QUIT;
for (i = 1; i <= 3; ++i)
{
  error = GRBresetmodel(model);
  if (error) goto QUIT;
  modelenv = GRBgetenv(model);
  if (!modelenv) {
    printf("Cannot retrieve model environment\n");
    exit(1);
  }
  error = GRBsetintparam(modelenv, "MIPFocus", i);
  if (error) goto QUIT;
  error = GRBoptimize(model);
  if (error) goto QUIT;
  error = GRBgetdblattr(model, "MIPGap", &gap);
  if (error) goto QUIT;
  if (bestgap > gap)
  {
    GRBmodel *tmp = bestmodel;
    bestmodel = model;
    model = tmp;
    bestgap = gap;
  }
}

/* Finally, free the extra model, reset the time limit and
   continue to solve the best model to optimality */
```

```
  GRBfreemodel(model);
  bestenv = GRBgetenv(bestmodel);
  if (!bestenv) {
    printf("Cannot retrieve best model environment\n");
    exit(1);
  }
  error = GRBsetdblparam(bestenv, "TimeLimit", GRB_INFINITY);
  if (error) goto QUIT;
  error = GRBoptimize(bestmodel);
  if (error) goto QUIT;
  error = GRBgetintparam(bestenv, "MIPFocus", &mipfocus);
  if (error) goto QUIT;

  printf("Solved with MIPFocus: %i\n", mipfocus);

QUIT:

  /* Error reporting */

  if (error)
  {
    printf("ERROR: %s\n", GRBgeterrormsg(env));
    exit(1);
  }

  /* Free best model */

  GRBfreemodel(bestmodel);

  /* Free environment */

  GRBfreeenv(env);

  return 0;
}
```

piecewise_c.c

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* This example considers the following separable, convex problem:

     minimize    f(x) - y + g(z)
     subject to  x + 2 y + 3 z <= 4
                 x +   y       >= 1
                 x,    y,    z <= 1

   where f(u) = exp(-u) and g(u) = 2 u^2 - 4 u, for all real u. It
   formulates and solves a simpler LP model by approximating f and
   g with piecewise-linear functions. Then it transforms the model
   into a MIP by negating the approximation for f, which corresponds
   to a non-convex piecewise-linear function, and solves it again.
*/

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "gurobi_c.h"

double f(double u) { return exp(-u); }
double g(double u) { return 2 * u * u - 4 * u; }

int
main(int   argc,
     char *argv[])
{
  GRBenv   *env   = NULL;
  GRBmodel *model = NULL;
  int       error = 0;
  double    lb, ub;
  int       npts, i;
  double   *ptu = NULL;
  double   *ptf = NULL;
  double   *ptg = NULL;
  int       ind[3];
  double    val[3];
  int       ismip;
  double    objval;
  double    sol[3];

  /* Create environment */
```

```
error = GRBloadenv(&env, NULL);
if (error) goto QUIT;

/* Create a new model */

error = GRBnewmodel(env, &model, NULL, 0, NULL, NULL, NULL, NULL, NULL);
if (error) goto QUIT;

/* Add variables */

lb = 0.0; ub = 1.0;

error = GRBaddvar(model, 0, NULL, NULL, 0.0, lb, ub, GRB_CONTINUOUS, "x");
if (error) goto QUIT;
error = GRBaddvar(model, 0, NULL, NULL, 0.0, lb, ub, GRB_CONTINUOUS, "y");
if (error) goto QUIT;
error = GRBaddvar(model, 0, NULL, NULL, 0.0, lb, ub, GRB_CONTINUOUS, "z");
if (error) goto QUIT;

/* Set objective for y */

error = GRBsetdblattrelement(model, GRB_DBL_ATTR_OBJ, 1, -1.0);
if (error) goto QUIT;

/* Add piecewise-linear objective functions for x and z */

npts = 101;
ptu = (double *) malloc(npts * sizeof(double));
ptf = (double *) malloc(npts * sizeof(double));
ptg = (double *) malloc(npts * sizeof(double));

for (i = 0; i < npts; i++) {
  ptu[i] = lb + (ub - lb) * i / (npts - 1);
  ptf[i] = f(ptu[i]);
  ptg[i] = g(ptu[i]);
}

error = GRBsetpwlobj(model, 0, npts, ptu, ptf);
if (error) goto QUIT;
error = GRBsetpwlobj(model, 2, npts, ptu, ptg);
if (error) goto QUIT;

/* Add constraint: x + 2 y + 3 z <= 4 */

ind[0] = 0; ind[1] = 1; ind[2] = 2;
```

```
val[0] = 1; val[1] = 2; val[2] = 3;

error = GRBaddconstr(model, 3, ind, val, GRB_LESS_EQUAL, 4.0, "c0");
if (error) goto QUIT;

/* Add constraint: x + y >= 1 */

ind[0] = 0; ind[1] = 1;
val[0] = 1; val[1] = 1;

error = GRBaddconstr(model, 2, ind, val, GRB_GREATER_EQUAL, 1.0, "c1");
if (error) goto QUIT;

/* Optimize model as an LP */

error = GRBoptimize(model);
if (error) goto QUIT;

error = GRBgetintattr(model, "IsMIP", &ismip);
if (error) goto QUIT;
error = GRBgetdblattr(model, "ObjVal", &objval);
if (error) goto QUIT;
error = GRBgetdblattrarray(model, "X", 0, 3, sol);
if (error) goto QUIT;

printf("IsMIP: %d\n", ismip);
printf("x %g\ny %g\nz %g\n", sol[0], sol[1], sol[2]);
printf("Obj: %g\n", objval);
printf("\n");

/* Negate piecewise-linear objective function for x */

for (i = 0; i < npts; i++) {
  ptf[i] = -ptf[i];
}

error = GRBsetpwlobj(model, 0, npts, ptu, ptf);
if (error) goto QUIT;

/* Optimize model as a MIP */

error = GRBoptimize(model);
if (error) goto QUIT;

error = GRBgetintattr(model, "IsMIP", &ismip);
```

```
  if (error) goto QUIT;
  error = GRBgetdblattr(model, "ObjVal", &objval);
  if (error) goto QUIT;
  error = GRBgetdblattrarray(model, "X", 0, 3, sol);
  if (error) goto QUIT;

  printf("IsMIP: %d\n", ismip);
  printf("x %g\ny %g\nz %g\n", sol[0], sol[1], sol[2]);
  printf("Obj: %g\n", objval);

QUIT:

  /* Error reporting */

  if (error) {
    printf("ERROR: %s\n", GRBgeterrormsg(env));
    exit(1);
  }

  /* Free data */

  free(ptu);
  free(ptf);
  free(ptg);

  /* Free model */

  GRBfreemodel(model);

  /* Free environment */

  GRBfreeenv(env);

  return 0;
}
```

poolsearch_c.c

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* We find alternative epsilon-optimal solutions to a given knapsack
 * problem by using PoolSearchMode */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "gurobi_c.h"

#define MAXSTR 128


int main(void)
{
  GRBenv   *env   = NULL;
  GRBenv   *menv  = NULL;
  GRBmodel *model = NULL;
  int       error = 0;
  char      buffer[MAXSTR];
  int e, status, nSolutions, prlen;
  double objval, *cval = NULL;
  int *cind = NULL;

  /* Sample data */
  const int groundSetSize = 10;
  double objCoef[10] =
    {32, 32, 15, 15, 6, 6, 1, 1, 1, 1};
  double knapsackCoef[10] =
    {16, 16,  8,  8, 4, 4, 2, 2, 1, 1};
  double Budget = 33;

  /* Create environment */
  error = GRBloadenv(&env, "poolsearch_c.log");
  if (error) goto QUIT;

  /* Create initial model */
  error = GRBnewmodel(env, &model, "poolsearch_c", groundSetSize, NULL,
                      NULL, NULL, NULL, NULL);
  if (error) goto QUIT;

  /* get model environment */
  menv = GRBgetenv(model);
```

```
if (!menv) {
  fprintf(stderr, "Error: could not get model environment\n");
  goto QUIT;
}

/* set objective function */
error = GRBsetdblattrarray(model, "Obj", 0, groundSetSize, objCoef);
if (error) goto QUIT;

/* set variable types and names */
for (e = 0; e < groundSetSize; e++) {
  sprintf(buffer, "El%d", e);
  error = GRBsetcharattrelement(model, "VType", e, GRB_BINARY);
  if (error) goto QUIT;

  error = GRBsetstrattrelement(model, "VarName", e, buffer);
  if (error) goto QUIT;
}

/* Make space for constraint data */
cind = malloc(sizeof(int) * groundSetSize);
if (!cind) goto QUIT;
for (e = 0; e < groundSetSize; e++)
  cind[e] = e;

/* Constraint: limit total number of elements to be picked to be at most
 * Budget */
sprintf (buffer, "Budget");
error = GRBaddconstr(model, groundSetSize, cind, knapsackCoef,
                     GRB_LESS_EQUAL, Budget, buffer);
if (error) goto QUIT;

/* set global sense for ALL objectives */
error = GRBsetintattr(model, GRB_INT_ATTR_MODELSENSE, GRB_MAXIMIZE);
if (error) goto QUIT;

/* Limit how many solutions to collect */
error = GRBsetintparam(menv, GRB_INT_PAR_POOLSOLUTIONS, 1024);
if (error) goto QUIT;

/* Limit the search space by setting a gap for the worst possible solution that will be accept
error = GRBsetdblparam(menv, GRB_DBL_PAR_POOLGAP, 0.10);
if (error) goto QUIT;

/* do a systematic search for the k-best solutions */
```

```
error = GRBsetintparam(menv, GRB_INT_PAR_POOLSEARCHMODE, 2);
if (error) goto QUIT;

/* save problem */
error = GRBwrite(model, "poolsearch_c.lp");
if (error) goto QUIT;
error = GRBwrite(model, "poolsearch_c.mps");
if (error) goto QUIT;

/* Optimize */
error = GRBoptimize(model);
if (error) goto QUIT;

/* Status checking */
error = GRBgetintattr(model, "Status", &status);
if (error) goto QUIT;

if (status == GRB_INF_OR_UNBD ||
    status == GRB_INFEASIBLE  ||
    status == GRB_UNBOUNDED     ) {
  printf("The model cannot be solved "
         "because it is infeasible or unbounded\n");
  goto QUIT;
}
if (status != GRB_OPTIMAL) {
  printf("Optimization was stopped with status %d\n", status);
  goto QUIT;
}

/* make space for optimal solution */
cval = malloc(sizeof(double) * groundSetSize);
if (!cval) goto QUIT;

/* Print best selected set */
error = GRBgetdblattrarray(model, GRB_DBL_ATTR_X, 0, groundSetSize, cval);
if (error) goto QUIT;

printf("Selected elements in best solution:\n\t");
for (e = 0; e < groundSetSize; e++) {
  if (cval[e] < .9) continue;
  printf("El%d ", e);
}

/* print number of solutions stored */
error = GRBgetintattr(model, GRB_INT_ATTR_SOLCOUNT, &nSolutions);
```

```
  if (error) goto QUIT;
  printf("\nNumber of solutions found: %d\nValues:", nSolutions);

  /* print objective values of alternative solutions */
  prlen = 0;
  for (e = 0; e < nSolutions; e++) {
    error = GRBsetintparam(menv, GRB_INT_PAR_SOLUTIONNUMBER, e);
    if (error) goto QUIT;

    error = GRBgetdblattr(model, GRB_DBL_ATTR_POOLOBJVAL, &objval);
    if (error) goto QUIT;

    prlen += printf(" %g", objval);
    if (prlen >= 75 && e+1 < nSolutions) {
      prlen = printf("\n    ");
    }
  }
  printf("\n");

  /* print fourth best set if available */
  if (nSolutions >= 4) {
    error = GRBsetintparam(menv, GRB_INT_PAR_SOLUTIONNUMBER, 3);
    if (error) goto QUIT;

    /* get the solution vector */
    error = GRBgetdblattrarray(model, GRB_DBL_ATTR_XN, 0, groundSetSize, cval);
    if (error) goto QUIT;

    printf("Selected elements in fourth best solution:\n\t");
    for (e = 0; e < groundSetSize; e++) {
      if (cval[e] < .9) continue;
      printf("El%d ", e);
    }
    printf("\n");
  }

QUIT:
  if (model != NULL) GRBfreemodel(model);
  if (env != NULL)   GRBfreeenv(env);
  if (cind != NULL)  free(cind);
  if (cval != NULL)  free(cval);
  return error;
}
```

qcp_c.c

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* This example formulates and solves the following simple QCP model:

     maximize    x
     subject to  x + y + z = 1
                 x^2 + y^2 <= z^2 (second-order cone)
                 x^2 <= yz        (rotated second-order cone)
*/

#include <stdlib.h>
#include <stdio.h>
#include "gurobi_c.h"

int
main(int   argc,
     char *argv[])
{
  GRBenv   *env   = NULL;
  GRBmodel *model = NULL;
  int       error = 0;
  double    sol[3];
  int       ind[3];
  double    val[3];
  double    obj[] = {1, 0, 0};
  int       qrow[3];
  int       qcol[3];
  double    qval[3];
  int       optimstatus;
  double    objval;

  /* Create environment */

  error = GRBloadenv(&env, "qcp.log");
  if (error) goto QUIT;

  /* Create an empty model */

  error = GRBnewmodel(env, &model, "qcp", 0, NULL, NULL, NULL, NULL, NULL);
  if (error) goto QUIT;


  /* Add variables */
```

```
error = GRBaddvars(model, 3, 0, NULL, NULL, NULL, obj, NULL, NULL, NULL,
                   NULL);
if (error) goto QUIT;

/* Change sense to maximization */

error = GRBsetintattr(model, GRB_INT_ATTR_MODELSENSE, GRB_MAXIMIZE);
if (error) goto QUIT;

/* Linear constraint: x + y + z = 1 */

ind[0] = 0; ind[1] = 1; ind[2] = 2;
val[0] = 1; val[1] = 1; val[2] = 1;

error = GRBaddconstr(model, 3, ind, val, GRB_EQUAL, 1.0, "c0");
if (error) goto QUIT;

/* Cone: x^2 + y^2 <= z^2 */

qrow[0] = 0; qcol[0] = 0; qval[0] = 1.0;
qrow[1] = 1; qcol[1] = 1; qval[1] = 1.0;
qrow[2] = 2; qcol[2] = 2; qval[2] = -1.0;

error = GRBaddqconstr(model, 0, NULL, NULL, 3, qrow, qcol, qval,
                      GRB_LESS_EQUAL, 0.0, "qc0");
if (error) goto QUIT;

/* Rotated cone: x^2 <= yz */

qrow[0] = 0; qcol[0] = 0; qval[0] = 1.0;
qrow[1] = 1; qcol[1] = 2; qval[1] = -1.0;

error = GRBaddqconstr(model, 0, NULL, NULL, 2, qrow, qcol, qval,
                      GRB_LESS_EQUAL, 0.0, "qc1");
if (error) goto QUIT;

/* Optimize model */

error = GRBoptimize(model);
if (error) goto QUIT;

/* Write model to 'qcp.lp' */

error = GRBwrite(model, "qcp.lp");
if (error) goto QUIT;
```

```
  /* Capture solution information */

  error = GRBgetintattr(model, GRB_INT_ATTR_STATUS, &optimstatus);
  if (error) goto QUIT;

  error = GRBgetdblattr(model, GRB_DBL_ATTR_OBJVAL, &objval);
  if (error) goto QUIT;

  error = GRBgetdblattrarray(model, GRB_DBL_ATTR_X, 0, 3, sol);
  if (error) goto QUIT;

  printf("\nOptimization complete\n");
  if (optimstatus == GRB_OPTIMAL) {
    printf("Optimal objective: %.4e\n", objval);

    printf("  x=%.2f, y=%.2f, z=%.2f\n", sol[0], sol[1], sol[2]);
  } else if (optimstatus == GRB_INF_OR_UNBD) {
    printf("Model is infeasible or unbounded\n");
  } else {
    printf("Optimization was stopped early\n");
  }

QUIT:

  /* Error reporting */

  if (error) {
    printf("ERROR: %s\n", GRBgeterrormsg(env));
    exit(1);
  }

  /* Free model */

  GRBfreemodel(model);

  /* Free environment */

  GRBfreeenv(env);

  return 0;
}
```

qp_c.c

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* This example formulates and solves the following simple QP model:

     minimize    x^2 + x*y + y^2 + y*z + z^2 + 2 x
     subject to  x + 2 y + 3 z >= 4
                 x +   y       >= 1

   It solves it once as a continuous model, and once as an integer model.
*/

#include <stdlib.h>
#include <stdio.h>
#include "gurobi_c.h"

int
main(int   argc,
     char *argv[])
{
  GRBenv   *env   = NULL;
  GRBmodel *model = NULL;
  int       error = 0;
  double    sol[3];
  int       ind[3];
  double    val[3];
  int       qrow[5];
  int       qcol[5];
  double    qval[5];
  char      vtype[3];
  int       optimstatus;
  double    objval;

  /* Create environment */

  error = GRBloadenv(&env, "qp.log");
  if (error) goto QUIT;

  /* Create an empty model */

  error = GRBnewmodel(env, &model, "qp", 0, NULL, NULL, NULL, NULL, NULL);
  if (error) goto QUIT;

  /* Add variables */
```

```
error = GRBaddvars(model, 3, 0, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
                   NULL);
if (error) goto QUIT;

/* Quadratic objective terms */

qrow[0] = 0; qrow[1] = 0; qrow[2] = 1; qrow[3] = 1; qrow[4] = 2;
qcol[0] = 0; qcol[1] = 1; qcol[2] = 1; qcol[3] = 2; qcol[4] = 2;
qval[0] = 1; qval[1] = 1; qval[2] = 1; qval[3] = 1; qval[4] = 1;

error = GRBaddqpterms(model, 5, qrow, qcol, qval);
if (error) goto QUIT;

/* Linear objective term */

error = GRBsetdblattrelement(model, GRB_DBL_ATTR_OBJ, 0, 2.0);
if (error) goto QUIT;

/* First constraint: x + 2 y + 3 z <= 4 */

ind[0] = 0; ind[1] = 1; ind[2] = 2;
val[0] = 1; val[1] = 2; val[2] = 3;

error = GRBaddconstr(model, 3, ind, val, GRB_GREATER_EQUAL, 4.0, "c0");
if (error) goto QUIT;

/* Second constraint: x + y >= 1 */

ind[0] = 0; ind[1] = 1;
val[0] = 1; val[1] = 1;

error = GRBaddconstr(model, 2, ind, val, GRB_GREATER_EQUAL, 1.0, "c1");
if (error) goto QUIT;

/* Optimize model */

error = GRBoptimize(model);
if (error) goto QUIT;

/* Write model to 'qp.lp' */

error = GRBwrite(model, "qp.lp");
if (error) goto QUIT;

/* Capture solution information */
```

```
error = GRBgetintattr(model, GRB_INT_ATTR_STATUS, &optimstatus);
if (error) goto QUIT;

error = GRBgetdblattr(model, GRB_DBL_ATTR_OBJVAL, &objval);
if (error) goto QUIT;

error = GRBgetdblattrarray(model, GRB_DBL_ATTR_X, 0, 3, sol);
if (error) goto QUIT;

printf("\nOptimization complete\n");
if (optimstatus == GRB_OPTIMAL) {
  printf("Optimal objective: %.4e\n", objval);

  printf("  x=%.4f, y=%.4f, z=%.4f\n", sol[0], sol[1], sol[2]);
} else if (optimstatus == GRB_INF_OR_UNBD) {
  printf("Model is infeasible or unbounded\n");
} else {
  printf("Optimization was stopped early\n");
}


/* Modify variable types */

vtype[0] = GRB_INTEGER; vtype[1] = GRB_INTEGER; vtype[2] = GRB_INTEGER;

error = GRBsetcharattrarray(model, GRB_CHAR_ATTR_VTYPE, 0, 3, vtype);
if (error) goto QUIT;

/* Optimize model */

error = GRBoptimize(model);
if (error) goto QUIT;

/* Write model to 'qp2.lp' */

error = GRBwrite(model, "qp2.lp");
if (error) goto QUIT;

/* Capture solution information */

error = GRBgetintattr(model, GRB_INT_ATTR_STATUS, &optimstatus);
if (error) goto QUIT;

error = GRBgetdblattr(model, GRB_DBL_ATTR_OBJVAL, &objval);
```

```
  if (error) goto QUIT;

  error = GRBgetdblattrarray(model, GRB_DBL_ATTR_X, 0, 3, sol);
  if (error) goto QUIT;

  printf("\nOptimization complete\n");
  if (optimstatus == GRB_OPTIMAL) {
    printf("Optimal objective: %.4e\n", objval);

    printf("  x=%.4f, y=%.4f, z=%.4f\n", sol[0], sol[1], sol[2]);
  } else if (optimstatus == GRB_INF_OR_UNBD) {
    printf("Model is infeasible or unbounded\n");
  } else {
    printf("Optimization was stopped early\n");
  }

QUIT:

  /* Error reporting */

  if (error) {
    printf("ERROR: %s\n", GRBgeterrormsg(env));
    exit(1);
  }

  /* Free model */

  GRBfreemodel(model);

  /* Free environment */

  GRBfreeenv(env);

  return 0;
}
```

sensitivity_c.c

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* A simple sensitivity analysis example which reads a MIP model
   from a file and solves it. Then each binary variable is set
   to 1-X, where X is its value in the optimal solution, and
   the impact on the objective function value is reported.
*/

#include <stdlib.h>
#include <stdio.h>
#include "gurobi_c.h"

int
main(int   argc,
     char *argv[])
{
  GRBenv   *env = NULL, *modelenv = NULL;
  GRBmodel *model = NULL;
  int       error = 0;
  int       ismip, status, numvars, i, j;
  double    origobjval, lb, ub, objval;
  double   *origx = NULL;
  char      vtype, *vname;

  if (argc < 2)
  {
    fprintf(stderr, "Usage: sensitivity_c filename\n");
    exit(1);
  }

  /* Create environment */

  error = GRBloadenv(&env, "sensitivity.log");
  if (error) goto QUIT;

  /* Read and solve model */

  error = GRBreadmodel(env, argv[1], &model);
  if (error) goto QUIT;

  error = GRBgetintattr(model, "IsMIP", &ismip);
  if (error) goto QUIT;
  if (ismip == 0) {
    printf("Model is not a MIP\n");
```

```
  exit(1);
}

error = GRBoptimize(model);
if (error) goto QUIT;

error = GRBgetintattr(model, "Status", &status);
if (error) goto QUIT;
if (status != GRB_OPTIMAL) {
  printf("Optimization ended with status %d\n", status);
  exit(1);
}

/* Store the optimal solution */

error = GRBgetdblattr(model, "ObjVal", &origobjval);
if (error) goto QUIT;
error = GRBgetintattr(model, "NumVars", &numvars);
if (error) goto QUIT;
origx = (double *) malloc(numvars * sizeof(double));
if (origx == NULL) {
  printf("Out of memory\n");
  exit(1);
}
error = GRBgetdblattrarray(model, "X", 0, numvars, origx);
if (error) goto QUIT;

/* Disable solver output for subsequent solves */

modelenv = GRBgetenv(model);
if (!modelenv) {
  printf("Cannot retrieve model environment\n");
  exit(1);
}
error = GRBsetintparam(modelenv, "OutputFlag", 0);
if (error) goto QUIT;

/* Iterate through unfixed, binary variables in model */

for (i = 0; i < numvars; i++) {
  error = GRBgetdblattrelement(model, "LB", i, &lb);
  if (error) goto QUIT;
  error = GRBgetdblattrelement(model, "UB", i, &ub);
  if (error) goto QUIT;
  error = GRBgetcharattrelement(model, "VType", i, &vtype);
```

```
if (error) goto QUIT;

if (lb == 0 && ub == 1
    && (vtype == GRB_BINARY || vtype == GRB_INTEGER)) {

  /* Set variable to 1-X, where X is its value in optimal solution */

  if (origx[i] < 0.5) {
    error = GRBsetdblattrelement(model, "LB", i, 1.0);
    if (error) goto QUIT;
    error = GRBsetdblattrelement(model, "Start", i, 1.0);
    if (error) goto QUIT;
  } else {
    error = GRBsetdblattrelement(model, "UB", i, 0.0);
    if (error) goto QUIT;
    error = GRBsetdblattrelement(model, "Start", i, 0.0);
    if (error) goto QUIT;
  }

  /* Update MIP start for the other variables */

  for (j = 0; j < numvars; j++) {
    if (j != i) {
      error = GRBsetdblattrelement(model, "Start", j, origx[j]);
      if (error) goto QUIT;
    }
  }

  /* Solve for new value and capture sensitivity information */

  error = GRBoptimize(model);
  if (error) goto QUIT;

  error = GRBgetintattr(model, "Status", &status);
  if (error) goto QUIT;
  error = GRBgetstrattrelement(model, "VarName", i, &vname);
  if (error) goto QUIT;
  if (status == GRB_OPTIMAL) {
    error = GRBgetdblattr(model, "ObjVal", &objval);
    if (error) goto QUIT;
    printf("Objective sensitivity for variable %s is %g\n",
        vname, objval - origobjval);
  } else {
    printf("Objective sensitivity for variable %s is infinite\n",
        vname);
```

```
      }

      /* Restore the original variable bounds */

      error = GRBsetdblattrelement(model, "LB", i, 0.0);
      if (error) goto QUIT;
      error = GRBsetdblattrelement(model, "UB", i, 1.0);
      if (error) goto QUIT;
    }
  }


QUIT:

  /* Error reporting */

  if (error)
  {
    printf("ERROR: %s\n", GRBgeterrormsg(env));
    exit(1);
  }

  /* Free data */

  free(origx);

  /* Free model */

  GRBfreemodel(model);

  /* Free environment */

  GRBfreeenv(env);

  return 0;
}
```

sos_c.c

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* This example creates a very simple Special Ordered Set (SOS) model.
   The model consists of 3 continuous variables, no linear constraints,
   and a pair of SOS constraints of type 1. */

#include <stdlib.h>
#include <stdio.h>
#include "gurobi_c.h"

int
main(int   argc,
     char *argv[])
{
  GRBenv   *env   = NULL;
  GRBmodel *model = NULL;
  int       error = 0;
  double    x[3];
  double    obj[3];
  double    ub[3];
  int       sostype[2];
  int       sosbeg[2];
  int       sosind[4];
  double    soswt[4];
  int       optimstatus;
  double    objval;

  /* Create environment */

  error = GRBloadenv(&env, "sos.log");
  if (error) goto QUIT;

  /* Create an empty model */

  error = GRBnewmodel(env, &model, "sos", 0, NULL, NULL, NULL, NULL, NULL);
  if (error) goto QUIT;


  /* Add variables */

  obj[0] = -2; obj[1] = -1; obj[2] = -1;
  ub[0] = 1.0; ub[1] = 1.0; ub[2] = 2.0;
  error = GRBaddvars(model, 3, 0, NULL, NULL, NULL, obj, NULL, ub, NULL,
                     NULL);
```

```
if (error) goto QUIT;

/* Build first SOS1: x0=0 or x1=0 */

sosind[0] = 0; sosind[1] = 1;
soswt[0] = 1.0; soswt[1] = 2.0;
sosbeg[0] = 0; sostype[0] = GRB_SOS_TYPE1;

/* Build second SOS1: x0=0 or x2=0 */

sosind[2] = 0; sosind[3] = 2;
soswt[2] = 1.0; soswt[3] = 2.0;
sosbeg[1] = 2; sostype[1] = GRB_SOS_TYPE1;

/* Add SOSs to model */

error = GRBaddsos(model, 2, 4, sostype, sosbeg, sosind, soswt);
if (error) goto QUIT;

/* Optimize model */

error = GRBoptimize(model);
if (error) goto QUIT;

/* Write model to 'sos.lp' */

error = GRBwrite(model, "sos.lp");
if (error) goto QUIT;

/* Capture solution information */

error = GRBgetintattr(model, GRB_INT_ATTR_STATUS, &optimstatus);
if (error) goto QUIT;

error = GRBgetdblattr(model, GRB_DBL_ATTR_OBJVAL, &objval);
if (error) goto QUIT;

error = GRBgetdblattrarray(model, GRB_DBL_ATTR_X, 0, 3, x);
if (error) goto QUIT;

printf("\nOptimization complete\n");
if (optimstatus == GRB_OPTIMAL) {
  printf("Optimal objective: %.4e\n", objval);

  printf("  x=%.4f, y=%.4f, z=%.4f\n", x[0], x[1], x[2]);
```

```
  } else if (optimstatus == GRB_INF_OR_UNBD) {
    printf("Model is infeasible or unbounded\n");
  } else {
    printf("Optimization was stopped early\n");
  }

QUIT:

  /* Error reporting */

  if (error) {
    printf("ERROR: %s\n", GRBgeterrormsg(env));
    exit(1);
  }

  /* Free model */

  GRBfreemodel(model);

  /* Free environment */

  GRBfreeenv(env);

  return 0;
}
```

sudoku_c.c

```
/* Copyright 2017, Gurobi Optimization, Inc. */
/*
  Sudoku example.

  The Sudoku board is a 9x9 grid, which is further divided into a 3x3 grid
  of 3x3 grids.  Each cell in the grid must take a value from 0 to 9.
  No two grid cells in the same row, column, or 3x3 subgrid may take the
  same value.

  In the MIP formulation, binary variables x[i,j,v] indicate whether
  cell <i,j> takes value 'v'.  The constraints are as follows:
    1. Each cell must take exactly one value (sum_v x[i,j,v] = 1)
    2. Each value is used exactly once per row (sum_i x[i,j,v] = 1)
    3. Each value is used exactly once per column (sum_j x[i,j,v] = 1)
    4. Each value is used exactly once per 3x3 subgrid (sum_grid x[i,j,v] = 1)

  Input datasets for this example can be found in examples/data/sudoku*.
*/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "gurobi_c.h"

#define SUBDIM  3
#define DIM    (SUBDIM*SUBDIM)

int
main(int   argc,
     char *argv[])
{
  FILE     *fp    = NULL;
  GRBenv   *env   = NULL;
  GRBmodel *model = NULL;
  int       board[DIM][DIM];
  char      inputline[100];
  int       ind[DIM];
  double    val[DIM];
  double    lb[DIM*DIM*DIM];
  char      vtype[DIM*DIM*DIM];
  char     *names[DIM*DIM*DIM];
  char      namestorage[10*DIM*DIM*DIM];
  char     *cursor;
  int       optimstatus;
```

```
double    objval;
int       zero = 0;
int       i, j, v, ig, jg, count;
int       error = 0;

if (argc < 2) {
  fprintf(stderr, "Usage: sudoku_c datafile\n");
  exit(1);
}

fp = fopen(argv[1], "r");
if (fp == NULL) {
  fprintf(stderr, "Error: unable to open input file %s\n", argv[1]);
  exit(1);
}

for (i = 0; i < DIM; i++) {
  fgets(inputline, 100, fp);
  if (strlen(inputline) < 9) {
    fprintf(stderr, "Error: not enough board positions specified\n");
    exit(1);
  }
  for (j = 0; j < DIM; j++) {
    board[i][j] = (int) inputline[j] - (int) '1';
    if (board[i][j] < 0 || board[i][j] >= DIM)
      board[i][j] = -1;
  }
}

/* Create an empty model */

cursor = namestorage;
for (i = 0; i < DIM; i++) {
  for (j = 0; j < DIM; j++) {
    for (v = 0; v < DIM; v++) {
      if (board[i][j] == v)
        lb[i*DIM*DIM+j*DIM+v] = 1;
      else
        lb[i*DIM*DIM+j*DIM+v] = 0;
      vtype[i*DIM*DIM+j*DIM+v] = GRB_BINARY;

      names[i*DIM*DIM+j*DIM+v] = cursor;
      sprintf(names[i*DIM*DIM+j*DIM+v], "x[%d,%d,%d]", i, j, v+1);
      cursor += strlen(names[i*DIM*DIM+j*DIM+v]) + 1;
    }
```

```
  }
}

/* Create environment */

error = GRBloadenv(&env, "sudoku.log");
if (error) goto QUIT;

/* Create new model */

error = GRBnewmodel(env, &model, "sudoku", DIM*DIM*DIM, NULL, lb, NULL,
                    vtype, names);
if (error) goto QUIT;

/* Each cell gets a value */

for (i = 0; i < DIM; i++) {
  for (j = 0; j < DIM; j++) {
    for (v = 0; v < DIM; v++) {
      ind[v] = i*DIM*DIM + j*DIM + v;
      val[v] = 1.0;
    }

    error = GRBaddconstr(model, DIM, ind, val, GRB_EQUAL, 1.0, NULL);
    if (error) goto QUIT;
  }
}

/* Each value must appear once in each row */

for (v = 0; v < DIM; v++) {
  for (j = 0; j < DIM; j++) {
    for (i = 0; i < DIM; i++) {
      ind[i] = i*DIM*DIM + j*DIM + v;
      val[i] = 1.0;
    }

    error = GRBaddconstr(model, DIM, ind, val, GRB_EQUAL, 1.0, NULL);
    if (error) goto QUIT;
  }
}

/* Each value must appear once in each column */

for (v = 0; v < DIM; v++) {
```

```
  for (i = 0; i < DIM; i++) {
    for (j = 0; j < DIM; j++) {
      ind[j] = i*DIM*DIM + j*DIM + v;
      val[j] = 1.0;
    }

    error = GRBaddconstr(model, DIM, ind, val, GRB_EQUAL, 1.0, NULL);
    if (error) goto QUIT;
  }
}

/* Each value must appear once in each subgrid */

for (v = 0; v < DIM; v++) {
  for (ig = 0; ig < SUBDIM; ig++) {
    for (jg = 0; jg < SUBDIM; jg++) {
      count = 0;
      for (i = ig*SUBDIM; i < (ig+1)*SUBDIM; i++) {
        for (j = jg*SUBDIM; j < (jg+1)*SUBDIM; j++) {
          ind[count] = i*DIM*DIM + j*DIM + v;
          val[count] = 1.0;
          count++;
        }
      }

      error = GRBaddconstr(model, DIM, ind, val, GRB_EQUAL, 1.0, NULL);
      if (error) goto QUIT;
    }
  }
}

/* Optimize model */

error = GRBoptimize(model);
if (error) goto QUIT;

/* Write model to 'sudoku.lp' */

error = GRBwrite(model, "sudoku.lp");
if (error) goto QUIT;

/* Capture solution information */

error = GRBgetintattr(model, GRB_INT_ATTR_STATUS, &optimstatus);
if (error) goto QUIT;
```

```
  error = GRBgetdblattr(model, GRB_DBL_ATTR_OBJVAL, &objval);
  if (error) goto QUIT;

  printf("\nOptimization complete\n");
  if (optimstatus == GRB_OPTIMAL)
    printf("Optimal objective: %.4e\n", objval);
  else if (optimstatus == GRB_INF_OR_UNBD)
    printf("Model is infeasible or unbounded\n");
  else
    printf("Optimization was stopped early\n");
  printf("\n");

QUIT:

  /* Error reporting */

  if (error) {
    printf("ERROR: %s\n", GRBgeterrormsg(env));
    exit(1);
  }

  /* Free model */

  GRBfreemodel(model);

  /* Free environment */

  GRBfreeenv(env);

  return 0;
}
```

tsp_c.c

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/*
  Solve a traveling salesman problem on a randomly generated set of
  points using lazy constraints.   The base MIP model only includes
  'degree-2' constraints, requiring each node to have exactly
  two incident edges.  Solutions to this model may contain subtours -
  tours that don't visit every node.  The lazy constraint callback
  adds new constraints to cut them off.
*/

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "gurobi_c.h"


/* Define structure to pass data to the callback function */

struct callback_data {
  int n;
};


/* Given an integer-feasible solution 'sol', find the smallest
   sub-tour.  Result is returned in 'tour', and length is
   returned in 'tourlenP'. */

static void
findsubtour(int     n,
            double *sol,
            int    *tourlenP,
            int    *tour)
{
  int i, node, len, start;
  int bestind, bestlen;
  int *seen = NULL;

  seen = (int *) malloc(n*sizeof(int));
  if (seen == NULL) {
    fprintf(stderr, "Out of memory\n");
    exit(1);
  }
```

```
  for (i = 0; i < n; i++)
    seen[i] = 0;

  start   = 0;
  bestlen = n+1;
  bestind = -1;
  while (start < n) {
    for (node = 0; node < n; node++)
      if (seen[node] == 0)
        break;
    if (node == n)
      break;
    for (len = 0; len < n; len++) {
      tour[start+len] = node;
      seen[node] = 1;
      for (i = 0; i < n; i++) {
        if (sol[node*n+i] > 0.5 && !seen[i]) {
          node = i;
          break;
        }
      }
      if (i == n) {
        len++;
        if (len < bestlen) {
          bestlen = len;
          bestind = start;
        }
        start += len;
        break;
      }
    }
  }

  for (i = 0; i < bestlen; i++)
    tour[i] = tour[bestind+i];
  *tourlenP = bestlen;

  free(seen);
}


/* Subtour elimination callback.  Whenever a feasible solution is found,
   find the shortest subtour, and add a subtour elimination constraint
   if that tour doesn't visit every node. */
```

```
int __stdcall
subtourelim(GRBmodel *model,
            void     *cbdata,
            int       where,
            void     *usrdata)
{
  struct callback_data *mydata = (struct callback_data *) usrdata;
  int n = mydata->n;
  int *tour = NULL;
  double *sol = NULL;
  int i, j, len, nz;
  int error = 0;

  if (where == GRB_CB_MIPSOL) {
    sol  = (double *) malloc(n*n*sizeof(double));
    tour = (int *)    malloc(n*sizeof(int));
    if (sol == NULL || tour == NULL) {
      fprintf(stderr, "Out of memory\n");
      exit(1);
    }

    GRBcbget(cbdata, where, GRB_CB_MIPSOL_SOL, sol);

    findsubtour(n, sol, &len, tour);

    if (len < n) {
      int    *ind = NULL;
      double *val = NULL;

      ind = (int *)    malloc(len*(len-1)/2*sizeof(int));
      val = (double *) malloc(len*(len-1)/2*sizeof(double));

      if (ind == NULL || val == NULL) {
        fprintf(stderr, "Out of memory\n");
        exit(1);
      }

      /* Add subtour elimination constraint */

      nz = 0;
      for (i = 0; i < len; i++)
        for (j = i+1; j < len; j++)
          ind[nz++] = tour[i]*n+tour[j];
      for (i = 0; i < nz; i++)
        val[i] = 1.0;
```

```
      error = GRBcblazy(cbdata, nz, ind, val, GRB_LESS_EQUAL, len-1);

      free(ind);
      free(val);
    }

    free(sol);
    free(tour);
  }

  return error;
}

/* Euclidean distance between points 'i' and 'j'. */

static double
distance(double *x,
         double *y,
         int     i,
         int     j)
{
  double dx = x[i] - x[j];
  double dy = y[i] - y[j];

  return sqrt(dx*dx + dy*dy);
}

int
main(int   argc,
     char *argv[])
{
  GRBenv   *env   = NULL;
  GRBmodel *model = NULL;
  int       i, j, len, n, solcount;
  int       error = 0;
  char      name[100];
  double   *x = NULL;
  double   *y = NULL;
  int      *ind = NULL;
  double   *val = NULL;
  struct callback_data mydata;

  if (argc < 2) {
    fprintf(stderr, "Usage: tsp_c size\n");
```

```
  exit(1);
}

n = atoi(argv[1]);
if (n == 0) {
  fprintf(stderr, "Argument must be a positive integer.\n");
} else if (n > 100) {
  printf("It will be a challenge to solve a TSP this large.\n");
}

x   = (double *) malloc(n*sizeof(double));
y   = (double *) malloc(n*sizeof(double));
ind = (int *)    malloc(n*sizeof(int));
val = (double *) malloc(n*sizeof(double));

if (x == NULL || y == NULL || ind == NULL || val == NULL) {
  fprintf(stderr, "Out of memory\n");
  exit(1);
}

/* Create random points */

for (i = 0; i < n; i++) {
  x[i] = ((double) rand())/RAND_MAX;
  y[i] = ((double) rand())/RAND_MAX;
}

/* Create environment */

error = GRBloadenv(&env, "tsp.log");
if (error) goto QUIT;

/* Create an empty model */

error = GRBnewmodel(env, &model, "tsp", 0, NULL, NULL, NULL, NULL, NULL);
if (error) goto QUIT;


/* Add variables - one for every pair of nodes */
/* Note: If edge from i to j is chosen, then x[i*n+j] = x[j*n+i] = 1. */
/* The cost is split between the two variables. */

for (i = 0; i < n; i++) {
  for (j = 0; j < n; j++) {
    sprintf(name, "x_%d_%d", i, j);
```

```
      error = GRBaddvar(model, 0, NULL, NULL, distance(x, y, i, j)/2,
                        0.0, 1.0, GRB_BINARY, name);
      if (error) goto QUIT;
    }
  }

  /* Degree-2 constraints */

  for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
      ind[j] = i*n+j;
      val[j] = 1.0;
    }

    sprintf(name, "deg2_%d", i);

    error = GRBaddconstr(model, n, ind, val, GRB_EQUAL, 2, name);
    if (error) goto QUIT;
  }

  /* Forbid edge from node back to itself */

  for (i = 0; i < n; i++) {
    error = GRBsetdblattrelement(model, GRB_DBL_ATTR_UB, i*n+i, 0);
    if (error) goto QUIT;
  }

  /* Symmetric TSP */

  for (i = 0; i < n; i++) {
    for (j = 0; j < i; j++) {
      ind[0] = i*n+j;
      ind[1] = i+j*n;
      val[0] = 1;
      val[1] = -1;
      error = GRBaddconstr(model, 2, ind, val, GRB_EQUAL, 0, NULL);
      if (error) goto QUIT;
    }
  }

  /* Set callback function */

  mydata.n = n;

  error = GRBsetcallbackfunc(model, subtourelim, (void *) &mydata);
```

```
  if (error) goto QUIT;

  /* Must set LazyConstraints parameter when using lazy constraints */

  error = GRBsetintparam(GRBgetenv(model), GRB_INT_PAR_LAZYCONSTRAINTS, 1);
  if (error) goto QUIT;

  /* Optimize model */

  error = GRBoptimize(model);
  if (error) goto QUIT;

  /* Extract solution */

  error = GRBgetintattr(model, GRB_INT_ATTR_SOLCOUNT, &solcount);
  if (error) goto QUIT;

  if (solcount > 0) {
    int *tour = NULL;
    double *sol = NULL;

    sol = (double *) malloc(n*n*sizeof(double));
    tour = (int *) malloc(n*sizeof(int));
    if (sol == NULL || tour == NULL) {
      fprintf(stderr, "Out of memory\n");
      exit(1);
    }

    error = GRBgetdblattrarray(model, GRB_DBL_ATTR_X, 0, n*n, sol);
    if (error) goto QUIT;

    /* Print tour */

    findsubtour(n, sol, &len, tour);

    printf("Tour: ");
    for (i = 0; i < len; i++)
      printf("%d ", tour[i]);
    printf("\n");

    free(tour);
    free(sol);
  }

QUIT:
```

```
  /* Free data */

  free(x);
  free(y);
  free(ind);
  free(val);

  /* Error reporting */

  if (error) {
    printf("ERROR: %s\n", GRBgeterrormsg(env));
    exit(1);
  }

  /* Free model */

  GRBfreemodel(model);

  /* Free environment */

  GRBfreeenv(env);

  return 0;
}
```

tune_c.c

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* This example reads a model from a file and tunes it.
   It then writes the best parameter settings to a file
   and solves the model using these parameters. */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "gurobi_c.h"

int
main(int   argc,
     char *argv[])
{
  GRBenv   *env   = NULL;
  GRBmodel *model = NULL;
  int       tuneresultcount;
  int       error = 0;

  if (argc < 2) {
    fprintf(stderr, "Usage: tune_c filename\n");
    exit(1);
  }

  /* Create environment */

  error = GRBloadenv(&env, "tune_c.log");
  if (error) goto QUIT;

  /* Read model from file */

  error = GRBreadmodel(env, argv[1], &model);
  if (error) goto QUIT;

  /* Set the TuneResults parameter to 1 */

  error = GRBsetintparam(GRBgetenv(model), GRB_INT_PAR_TUNERESULTS, 1);
  if (error) goto QUIT;

  /* Tune the model */

  error = GRBtunemodel(model);
  if (error) goto QUIT;
```

```
  /* Get the number of tuning results */

  error = GRBgetintattr(model, GRB_INT_ATTR_TUNE_RESULTCOUNT, &tuneresultcount);
  if (error) goto QUIT;

  if (tuneresultcount > 0) {

    /* Load the best tuned parameters into the model's environment */

    error = GRBgettuneresult(model, 0);
    if (error) goto QUIT;

    /* Write tuned parameters to a file */

    error = GRBwrite(model, "tune.prm");
    if (error) goto QUIT;

    /* Solve the model using the tuned parameters */

    error = GRBoptimize(model);
    if (error) goto QUIT;
  }

QUIT:

  /* Error reporting */

  if (error) {
    printf("ERROR: %s\n", GRBgeterrormsg(env));
    exit(1);
  }

  /* Free model */

  GRBfreemodel(model);

  /* Free environment */

  GRBfreeenv(env);

  return 0;
}
```

workforce1_c.c

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* Assign workers to shifts; each worker may or may not be available on a
   particular day. If the problem cannot be solved, use IIS to find a set of
   conflicting constraints. Note that there may be additional conflicts
   besides what is reported via IIS. */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "gurobi_c.h"


#define xcol(w,s)  nShifts*w+s
#define MAXSTR     128


int
main(int   argc,
     char *argv[])
{
  GRBenv   *env   = NULL;
  GRBmodel *model = NULL;
  int       error = 0, status;
  int       s, w, col;
  int      *cbeg = NULL;
  int      *cind = NULL;
  int       idx;
  double   *cval = NULL;
  char     *sense = NULL;
  char      vname[MAXSTR];
  double    obj;
  int       i, iis, numconstrs;
  char     *cname;

  /* Sample data */
  const int nShifts = 14;
  const int nWorkers = 7;

  /* Sets of days and workers */
  char* Shifts[] =
    { "Mon1", "Tue2", "Wed3", "Thu4", "Fri5", "Sat6",
      "Sun7", "Mon8", "Tue9", "Wed10", "Thu11", "Fri12", "Sat13",
      "Sun14" };
```

```
char* Workers[] =
  { "Amy", "Bob", "Cathy", "Dan", "Ed", "Fred", "Gu" };

/* Number of workers required for each shift */
double shiftRequirements[] =
  { 3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5 };

/* Amount each worker is paid to work one shift */
double pay[] = { 10, 12, 10, 8, 8, 9, 11 };

/* Worker availability: 0 if the worker is unavailable for a shift */
double availability[][14] =
  { { 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1 },
    { 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0 },
    { 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
    { 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1 },
    { 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1 },
    { 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1 },
    { 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 } };

/* Create environment */
error = GRBloadenv(&env, "workforce1.log");
if (error) goto QUIT;

/* Create initial model */
error = GRBnewmodel(env, &model, "workforce1", nWorkers * nShifts,
                    NULL, NULL, NULL, NULL, NULL);
if (error) goto QUIT;

/* Initialize assignment decision variables:
   x[w][s] == 1 if worker w is assigned
   to shift s. Since an assignment model always produces integer
   solutions, we use continuous variables and solve as an LP. */
for (w = 0; w < nWorkers; ++w)
{
  for (s = 0; s < nShifts; ++s)
  {
    col = xcol(w, s);
    sprintf(vname, "%s.%s", Workers[w], Shifts[s]);
    error = GRBsetdblattrelement(model, "UB", col, availability[w][s]);
    if (error) goto QUIT;
    error = GRBsetdblattrelement(model, "Obj", col, pay[w]);
    if (error) goto QUIT;
    error = GRBsetstrattrelement(model, "VarName", col, vname);
    if (error) goto QUIT;
```

```
  }
}

/* The objective is to minimize the total pay costs */
error = GRBsetintattr(model, "ModelSense", GRB_MINIMIZE);
if (error) goto QUIT;

/* Make space for constraint data */
cbeg = malloc(sizeof(int) * nShifts);
if (!cbeg) goto QUIT;
cind = malloc(sizeof(int) * nShifts * nWorkers);
if (!cind) goto QUIT;
cval = malloc(sizeof(double) * nShifts * nWorkers);
if (!cval) goto QUIT;
sense = malloc(sizeof(char) * nShifts);
if (!sense) goto QUIT;

/* Constraint: assign exactly shiftRequirements[s] workers
   to each shift s */
idx = 0;
for (s = 0; s < nShifts; ++s)
{
  cbeg[s] = idx;
  sense[s] = GRB_EQUAL;
  for (w = 0; w < nWorkers; ++w)
  {
    cind[idx] = xcol(w, s);
    cval[idx++] = 1.0;
  }
}
error = GRBaddconstrs(model, nShifts, idx, cbeg, cind, cval, sense,
                      shiftRequirements, Shifts);
if (error) goto QUIT;

/* Optimize */
error = GRBoptimize(model);
if (error) goto QUIT;
error = GRBgetintattr(model, "Status", &status);
if (error) goto QUIT;
if (status == GRB_UNBOUNDED)
{
  printf("The model cannot be solved because it is unbounded\n");
  goto QUIT;
}
if (status == GRB_OPTIMAL)
```

```
  {
    error = GRBgetdblattr(model, "ObjVal", &obj);
    if (error) goto QUIT;
    printf("The optimal objective is %f\n", obj);
    goto QUIT;
  }
  if ((status != GRB_INF_OR_UNBD) && (status != GRB_INFEASIBLE))
  {
    printf("Optimization was stopped with status %i\n", status);
    goto QUIT;
  }

  /* do IIS */
  printf("The model is infeasible; computing IIS\n");
  error = GRBcomputeIIS(model);
  if (error) goto QUIT;
  printf("\nThe following constraint(s) cannot be satisfied:\n");
  error = GRBgetintattr(model, "NumConstrs", &numconstrs);
  if (error) goto QUIT;
  for (i = 0; i < numconstrs; ++i)
  {
    error = GRBgetintattrelement(model, "IISConstr", i, &iis);
    if (error) goto QUIT;
    if (iis)
    {
      error = GRBgetstrattrelement(model, "ConstrName", i, &cname);
      if (error) goto QUIT;
      printf("%s\n", cname);
    }
  }



QUIT:

  /* Error reporting */

  if (error)
  {
    printf("ERROR: %s\n", GRBgeterrormsg(env));
    exit(1);
  }

  /* Free data */
```

```
  free(cbeg);
  free(cind);
  free(cval);
  free(sense);

  /* Free model */

  GRBfreemodel(model);

  /* Free environment */

  GRBfreeenv(env);

  return 0;
}
```

workforce2_c.c

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* Assign workers to shifts; each worker may or may not be available on a
   particular day. If the problem cannot be solved, use IIS iteratively to
   find all conflicting constraints. */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "gurobi_c.h"


#define xcol(w,s)  nShifts*w+s
#define MAXSTR     128


int
main(int   argc,
     char *argv[])
{
  GRBenv   *env   = NULL;
  GRBmodel *model = NULL;
  int       error = 0, status;
  int       s, w, col;
  int      *cbeg = NULL;
  int      *cind = NULL;
  int       idx;
  double   *cval = NULL;
  char     *sense = NULL;
  char      vname[MAXSTR];
  double    obj;
  int       i, iis, numconstrs, numremoved = 0;
  char     *cname;
  char    **removed = NULL;

  /* Sample data */
  const int nShifts = 14;
  const int nWorkers = 7;

  /* Sets of days and workers */
  char* Shifts[] =
    { "Mon1", "Tue2", "Wed3", "Thu4", "Fri5", "Sat6",
      "Sun7", "Mon8", "Tue9", "Wed10", "Thu11", "Fri12", "Sat13",
```

```
    "Sun14" };
char* Workers[] =
  { "Amy", "Bob", "Cathy", "Dan", "Ed", "Fred", "Gu" };

/* Number of workers required for each shift */
double shiftRequirements[] =
  { 3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5 };

/* Amount each worker is paid to work one shift */
double pay[] = { 10, 12, 10, 8, 8, 9, 11 };

/* Worker availability: 0 if the worker is unavailable for a shift */
double availability[][14] =
  { { 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1 },
    { 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0 },
    { 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
    { 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1 },
    { 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1 },
    { 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1 },
    { 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 } };

/* Create environment */
error = GRBloadenv(&env, "workforce2.log");
if (error) goto QUIT;

/* Create initial model */
error = GRBnewmodel(env, &model, "workforce2", nWorkers * nShifts,
                    NULL, NULL, NULL, NULL, NULL);
if (error) goto QUIT;

/* Initialize assignment decision variables:
   x[w][s] == 1 if worker w is assigned
   to shift s. Since an assignment model always produces integer
   solutions, we use continuous variables and solve as an LP. */
for (w = 0; w < nWorkers; ++w)
{
  for (s = 0; s < nShifts; ++s)
  {
    col = xcol(w, s);
    sprintf(vname, "%s.%s", Workers[w], Shifts[s]);
    error = GRBsetdblattrelement(model, "UB", col, availability[w][s]);
    if (error) goto QUIT;
    error = GRBsetdblattrelement(model, "Obj", col, pay[w]);
    if (error) goto QUIT;
    error = GRBsetstrattrelement(model, "VarName", col, vname);
```

```
    if (error) goto QUIT;
  }
}

/* The objective is to minimize the total pay costs */
error = GRBsetintattr(model, "ModelSense", GRB_MINIMIZE);
if (error) goto QUIT;

/* Make space for constraint data */
cbeg = malloc(sizeof(int) * nShifts);
if (!cbeg) goto QUIT;
cind = malloc(sizeof(int) * nShifts * nWorkers);
if (!cind) goto QUIT;
cval = malloc(sizeof(double) * nShifts * nWorkers);
if (!cval) goto QUIT;
sense = malloc(sizeof(char) * nShifts);
if (!sense) goto QUIT;

/* Constraint: assign exactly shiftRequirements[s] workers
   to each shift s */
idx = 0;
for (s = 0; s < nShifts; ++s)
{
  cbeg[s] = idx;
  sense[s] = GRB_EQUAL;
  for (w = 0; w < nWorkers; ++w)
  {
    cind[idx] = xcol(w, s);
    cval[idx++] = 1.0;
  }
}
error = GRBaddconstrs(model, nShifts, idx, cbeg, cind, cval, sense,
                      shiftRequirements, Shifts);
if (error) goto QUIT;

/* Optimize */
error = GRBoptimize(model);
if (error) goto QUIT;
error = GRBgetintattr(model, "Status", &status);
if (error) goto QUIT;
if (status == GRB_UNBOUNDED)
{
  printf("The model cannot be solved because it is unbounded\n");
  goto QUIT;
}
```

```
if (status == GRB_OPTIMAL)
{
  error = GRBgetdblattr(model, "ObjVal", &obj);
  if (error) goto QUIT;
  printf("The optimal objective is %f\n", obj);
  goto QUIT;
}
if ((status != GRB_INF_OR_UNBD) && (status != GRB_INFEASIBLE))
{
  printf("Optimization was stopped with status %i\n", status);
  goto QUIT;
}

/* do IIS */
printf("The model is infeasible; computing IIS\n");

/* Loop until we reduce to a model that can be solved */
error = GRBgetintattr(model, "NumConstrs", &numconstrs);
if (error) goto QUIT;
removed = calloc(numconstrs, sizeof(char*));
if (!removed) goto QUIT;
while (1)
{
  error = GRBcomputeIIS(model);
  if (error) goto QUIT;
  printf("\nThe following constraint cannot be satisfied:\n");
  for (i = 0; i < numconstrs; ++i)
  {
    error = GRBgetintattrelement(model, "IISConstr", i, &iis);
    if (error) goto QUIT;
    if (iis)
    {
      error = GRBgetstrattrelement(model, "ConstrName", i, &cname);
      if (error) goto QUIT;
      printf("%s\n", cname);
      /* Remove a single constraint from the model */
      removed[numremoved] = malloc(sizeof(char) * (1+strlen(cname)));
      if (!removed[numremoved]) goto QUIT;
      strcpy(removed[numremoved++], cname);
      cind[0] = i;
      error = GRBdelconstrs(model, 1, cind);
      if (error) goto QUIT;
      break;
    }
  }
```

```
    printf("\n");
    error = GRBoptimize(model);
    if (error) goto QUIT;
    error = GRBgetintattr(model, "Status", &status);
    if (error) goto QUIT;
    if (status == GRB_UNBOUNDED)
    {
      printf("The model cannot be solved because it is unbounded\n");
      goto QUIT;
    }
    if (status == GRB_OPTIMAL)
    {
      break;
    }
    if ((status != GRB_INF_OR_UNBD) && (status != GRB_INFEASIBLE))
    {
      printf("Optimization was stopped with status %i\n", status);
      goto QUIT;
    }
  }

  printf("\nThe following constraints were removed to get a feasible LP:\n");
  for (i = 0; i < numremoved; ++i)
  {
    printf("%s ", removed[i]);
  }
  printf("\n");



QUIT:

  /* Error reporting */

  if (error)
  {
    printf("ERROR: %s\n", GRBgeterrormsg(env));
    exit(1);
  }

  /* Free data */

  free(cbeg);
  free(cind);
```

```
  free(cval);
  free(sense);
  for (i=0; i<numremoved; ++i)
  {
    free(removed[i]);
  }
  free(removed);

  /* Free model */

  GRBfreemodel(model);

  /* Free environment */

  GRBfreeenv(env);

  return 0;
}
```

workforce3_c.c

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* Assign workers to shifts; each worker may or may not be available on a
   particular day. If the problem cannot be solved, relax the model
   to determine which constraints cannot be satisfied, and how much
   they need to be relaxed. */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "gurobi_c.h"


#define xcol(w,s)  nShifts*w+s
#define MAXSTR     128


int
main(int   argc,
     char *argv[])
{
  GRBenv   *env = NULL;
  GRBmodel *model = NULL;
  int       error = 0, status;
  int       s, w, col;
  int      *cbeg = NULL;
  int      *cind = NULL;
  int       idx;
  double   *cval = NULL;
  char     *sense = NULL;
  char      vname[MAXSTR];
  double    obj;
  int       i, j, orignumvars, numvars, numconstrs;
  double   *rhspen = NULL;
  double    sol;
  char     *sname;

  /* Sample data */
  const int nShifts = 14;
  const int nWorkers = 7;

  /* Sets of days and workers */
  char* Shifts[] =
```

```
  { "Mon1", "Tue2", "Wed3", "Thu4", "Fri5", "Sat6",
    "Sun7", "Mon8", "Tue9", "Wed10", "Thu11", "Fri12", "Sat13",
    "Sun14" };
char* Workers[] =
  { "Amy", "Bob", "Cathy", "Dan", "Ed", "Fred", "Gu" };

/* Number of workers required for each shift */
double shiftRequirements[] =
  { 3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5 };

/* Amount each worker is paid to work one shift */
double pay[] = { 10, 12, 10, 8, 8, 9, 11 };

/* Worker availability: 0 if the worker is unavailable for a shift */
double availability[][14] =
  { { 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1 },
    { 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0 },
    { 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
    { 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1 },
    { 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1 },
    { 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1 },
    { 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 } };

/* Create environment */
error = GRBloadenv(&env, "workforce3.log");
if (error) goto QUIT;

/* Create initial model */
error = GRBnewmodel(env, &model, "workforce3", nWorkers * nShifts,
                    NULL, NULL, NULL, NULL, NULL);
if (error) goto QUIT;

/* Initialize assignment decision variables:
   x[w][s] == 1 if worker w is assigned
   to shift s. Since an assignment model always produces integer
   solutions, we use continuous variables and solve as an LP. */
for (w = 0; w < nWorkers; ++w)
{
  for (s = 0; s < nShifts; ++s)
  {
    col = xcol(w, s);
    sprintf(vname, "%s.%s", Workers[w], Shifts[s]);
    error = GRBsetdblattrelement(model, "UB", col, availability[w][s]);
    if (error) goto QUIT;
    error = GRBsetdblattrelement(model, "Obj", col, pay[w]);
```

```
    if (error) goto QUIT;
    error = GRBsetstrattrelement(model, "VarName", col, vname);
    if (error) goto QUIT;
  }
}

/* The objective is to minimize the total pay costs */
error = GRBsetintattr(model, "ModelSense", GRB_MINIMIZE);
if (error) goto QUIT;

/* Make space for constraint data */
cbeg = malloc(sizeof(int) * nShifts);
if (!cbeg) goto QUIT;
cind = malloc(sizeof(int) * nShifts * nWorkers);
if (!cind) goto QUIT;
cval = malloc(sizeof(double) * nShifts * nWorkers);
if (!cval) goto QUIT;
sense = malloc(sizeof(char) * nShifts);
if (!sense) goto QUIT;

/* Constraint: assign exactly shiftRequirements[s] workers
   to each shift s */
idx = 0;
for (s = 0; s < nShifts; ++s)
{
  cbeg[s] = idx;
  sense[s] = GRB_EQUAL;
  for (w = 0; w < nWorkers; ++w)
  {
    cind[idx] = xcol(w, s);
    cval[idx++] = 1.0;
  }
}
error = GRBaddconstrs(model, nShifts, idx, cbeg, cind, cval, sense,
                      shiftRequirements, Shifts);
if (error) goto QUIT;

/* Optimize */
error = GRBoptimize(model);
if (error) goto QUIT;
error = GRBgetintattr(model, "Status", &status);
if (error) goto QUIT;
if (status == GRB_UNBOUNDED)
{
  printf("The model cannot be solved because it is unbounded\n");
```

```
  goto QUIT;
}
if (status == GRB_OPTIMAL)
{
  error = GRBgetdblattr(model, "ObjVal", &obj);
  if (error) goto QUIT;
  printf("The optimal objective is %f\n", obj);
  goto QUIT;
}
if ((status != GRB_INF_OR_UNBD) && (status != GRB_INFEASIBLE))
{
  printf("Optimization was stopped with status %i\n", status);
  goto QUIT;
}

/* Relax the constraints to make the model feasible */
printf("The model is infeasible; relaxing the constraints\n");

/* Determine the matrix size before relaxing the constraints */
error = GRBgetintattr(model, "NumVars", &orignumvars);
if (error) goto QUIT;
error = GRBgetintattr(model, "NumConstrs", &numconstrs);
if (error) goto QUIT;

/* Use FeasRelax feature with penalties for constraint violations */
rhspen = malloc(sizeof(double) * numconstrs);
if (!rhspen) goto QUIT;
for (i = 0; i < numconstrs; i++) rhspen[i] = 1;
error = GRBfeasrelax(model, GRB_FEASRELAX_LINEAR, 0,
                     NULL, NULL, rhspen, NULL);
if (error) goto QUIT;
error = GRBoptimize(model);
if (error) goto QUIT;
error = GRBgetintattr(model, "Status", &status);
if (error) goto QUIT;
if ((status == GRB_INF_OR_UNBD) || (status == GRB_INFEASIBLE) ||
    (status == GRB_UNBOUNDED))
{
  printf("The relaxed model cannot be solved "
         "because it is infeasible or unbounded\n");
  goto QUIT;
}
if (status != GRB_OPTIMAL)
{
  printf("Optimization was stopped with status %i\n", status);
```

```
    goto QUIT;
  }

  printf("\nSlack values:\n");
  error = GRBgetintattr(model, "NumVars", &numvars);
  if (error) goto QUIT;
  for (j = orignumvars; j < numvars; ++j)
  {
    error = GRBgetdblattrelement(model, "X", j, &sol);
    if (error) goto QUIT;
    if (sol > 1e-6)
    {
      error = GRBgetstrattrelement(model, "VarName", j, &sname);
      if (error) goto QUIT;
      printf("%s = %f\n", sname, sol);
    }
  }

QUIT:

  /* Error reporting */

  if (error)
  {
    printf("ERROR: %s\n", GRBgeterrormsg(env));
    exit(1);
  }

  /* Free data */

  free(cbeg);
  free(cind);
  free(cval);
  free(sense);
  free(rhspen);

  /* Free model */

  GRBfreemodel(model);

  /* Free environment */

  GRBfreeenv(env);
```

```
  return 0;
}
```

workforce4_c.c

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* Assign workers to shifts; each worker may or may not be available on a
   particular day. We use Pareto optimization to solve the model:
   first, we minimize the linear sum of the slacks. Then, we constrain
   the sum of the slacks, and we minimize a quadratic objective that
   tries to balance the workload among the workers. */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "gurobi_c.h"

int solveAndPrint(GRBmodel* model,
                  int nShifts, int nWorkers, char** Workers,
                  int* status);


#define xcol(w,s)         nShifts*w+s
#define slackcol(s)       nShifts*nWorkers+s
#define totSlackcol       nShifts*(nWorkers+1)
#define totShiftscol(w)   nShifts*(nWorkers+1)+1+w
#define avgShiftscol      (nShifts+1)*(nWorkers+1)
#define diffShiftscol(w)  (nShifts+1)*(nWorkers+1)+1+w
#define MAXSTR     128


int
main(int   argc,
     char *argv[])
{
  GRBenv   *env = NULL;
  GRBmodel *model = NULL;
  int       error = 0, status;
  int       s, w, col;
  int      *cbeg = NULL;
  int      *cind = NULL;
  int       idx;
  double   *cval = NULL;
  char     *sense = NULL;
  char      vname[MAXSTR], cname[MAXSTR];
  double    val;
```

```
/* Sample data */
const int nShifts = 14;
const int nWorkers = 7;

/* Sets of days and workers */
char* Shifts[] =
  { "Mon1", "Tue2", "Wed3", "Thu4", "Fri5", "Sat6",
    "Sun7", "Mon8", "Tue9", "Wed10", "Thu11", "Fri12", "Sat13",
    "Sun14" };
char* Workers[] =
  { "Amy", "Bob", "Cathy", "Dan", "Ed", "Fred", "Gu" };

/* Number of workers required for each shift */
double shiftRequirements[] =
  { 3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5 };

/* Worker availability: 0 if the worker is unavailable for a shift */
double availability[][14] =
  { { 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1 },
    { 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0 },
    { 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
    { 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1 },
    { 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1 },
    { 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1 },
    { 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 } };

/* Create environment */
error = GRBloadenv(&env, "workforce4.log");
if (error) goto QUIT;

/* Create initial model */
error = GRBnewmodel(env, &model, "workforce4",
                    (nShifts + 1) * (nWorkers + 1),
                    NULL, NULL, NULL, NULL, NULL);
if (error) goto QUIT;

/* Initialize assignment decision variables:
   x[w][s] == 1 if worker w is assigned to shift s.
   This is no longer a pure assignment model, so we must
   use binary variables. */
for (w = 0; w < nWorkers; ++w)
{
  for (s = 0; s < nShifts; ++s)
  {
    col = xcol(w, s);
```

```
    sprintf(vname, "%s.%s", Workers[w], Shifts[s]);
    error = GRBsetcharattrelement(model, "VType", col, GRB_BINARY);
    if (error) goto QUIT;
    error = GRBsetdblattrelement(model, "UB", col, availability[w][s]);
    if (error) goto QUIT;
    error = GRBsetstrattrelement(model, "VarName", col, vname);
    if (error) goto QUIT;
  }
}

/* Initialize slack decision variables */
for (s = 0; s < nShifts; ++s)
{
  sprintf(vname, "%sSlack", Shifts[s]);
  error = GRBsetstrattrelement(model, "VarName", slackcol(s), vname);
  if (error) goto QUIT;
}

/* Initialize total slack decision variable */
error = GRBsetstrattrelement(model, "VarName", totSlackcol, "totSlack");
if (error) goto QUIT;

/* Initialize variables to count the total shifts worked by each worker */
for (w = 0; w < nWorkers; ++w)
{
  sprintf(vname, "%sTotShifts", Workers[w]);
  error = GRBsetstrattrelement(model, "VarName", totShiftscol(w), vname);
  if (error) goto QUIT;
}

/* The objective is to minimize the sum of the slacks */
error = GRBsetintattr(model, "ModelSense", GRB_MINIMIZE);
if (error) goto QUIT;
error = GRBsetdblattrelement(model, "Obj", totSlackcol, 1.0);
if (error) goto QUIT;

/* Make space for constraint data */
cbeg = malloc(sizeof(int) * nShifts);
if (!cbeg) goto QUIT;
cind = malloc(sizeof(int) * nShifts * (nWorkers + 1));
if (!cind) goto QUIT;
cval = malloc(sizeof(double) * nShifts * (nWorkers + 1));
if (!cval) goto QUIT;
sense = malloc(sizeof(char) * nShifts);
if (!sense) goto QUIT;
```

```
/* Constraint: assign exactly shiftRequirements[s] workers
   to each shift s, plus the slack */
idx = 0;
for (s = 0; s < nShifts; ++s)
{
  cbeg[s] = idx;
  sense[s] = GRB_EQUAL;
  for (w = 0; w < nWorkers; ++w)
  {
    cind[idx] = xcol(w, s);
    cval[idx++] = 1.0;
  }
  cind[idx] = slackcol(s);
  cval[idx++] = 1.0;
}
error = GRBaddconstrs(model, nShifts, idx, cbeg, cind, cval, sense,
                      shiftRequirements, Shifts);
if (error) goto QUIT;

/* Constraint: set totSlack column equal to the total slack */
idx = 0;
for (s = 0; s < nShifts; ++s)
{
  cind[idx] = slackcol(s);
  cval[idx++] = 1.0;
}
cind[idx] = totSlackcol;
cval[idx++] = -1.0;
error = GRBaddconstr(model, idx, cind, cval, GRB_EQUAL,
                     0.0, "totSlack");
if (error) goto QUIT;

/* Constraint: compute the total number of shifts for each worker */
for (w = 0; w < nWorkers; ++w)
{
  idx = 0;
  for (s = 0; s < nShifts; ++s)
  {
    cind[idx] = xcol(w,s);
    cval[idx++] = 1.0;
  }
  sprintf(cname, "totShifts%s", Workers[w]);
  cind[idx] = totShiftscol(w);
  cval[idx++] = -1.0;
```

```
  error = GRBaddconstr(model, idx, cind, cval, GRB_EQUAL, 0.0, cname);
  if (error) goto QUIT;
}

/* Optimize */
error = solveAndPrint(model, nShifts, nWorkers, Workers, &status);
if (error) goto QUIT;
if (status != GRB_OPTIMAL) goto QUIT;

/* Constrain the slack by setting its upper and lower bounds */
error = GRBgetdblattrelement(model, "X", totSlackcol, &val);
if (error) goto QUIT;
error = GRBsetdblattrelement(model, "UB", totSlackcol, val);
if (error) goto QUIT;
error = GRBsetdblattrelement(model, "LB", totSlackcol, val);
if (error) goto QUIT;

/* Variable to count the average number of shifts worked */
error = GRBaddvar(model, 0, NULL, NULL, 0, 0, GRB_INFINITY, GRB_CONTINUOUS,
                  "avgShifts");
if (error) goto QUIT;

/* Variables to count the difference from average for each worker;
   note that these variables can take negative values. */
error = GRBaddvars(model, nWorkers, 0, NULL, NULL, NULL, NULL, NULL, NULL,
                   NULL, NULL);
if (error) goto QUIT;

for (w = 0; w < nWorkers; ++w)
{
  sprintf(vname, "%sDiff", Workers[w]);
  error = GRBsetstrattrelement(model, "VarName", diffShiftscol(w), vname);
  if (error) goto QUIT;
  error = GRBsetdblattrelement(model, "LB", diffShiftscol(w), -GRB_INFINITY);
  if (error) goto QUIT;
}

/* Constraint: compute the average number of shifts worked */
idx = 0;
for (w = 0; w < nWorkers; ++w)
{
  cind[idx] = totShiftscol(w);
  cval[idx++] = 1.0;
}
cind[idx] = avgShiftscol;
```

```
  cval[idx++] = -nWorkers;
  error = GRBaddconstr(model, idx, cind, cval, GRB_EQUAL, 0.0, "avgShifts");
  if (error) goto QUIT;

  /* Constraint: compute the difference from the average number of shifts */
  for (w = 0; w < nWorkers; ++w)
  {
    cind[0] = totShiftscol(w);
    cval[0] = 1.0;
    cind[1] = avgShiftscol;
    cval[1] = -1.0;
    cind[2] = diffShiftscol(w);
    cval[2] = -1.0;
    sprintf(cname, "%sDiff", Workers[w]);
    error = GRBaddconstr(model, 3, cind, cval, GRB_EQUAL, 0.0, cname);
    if (error) goto QUIT;
  }

  /* Objective: minimize the sum of the square of the difference from the
     average number of shifts worked */
  error = GRBsetdblattrelement(model, "Obj", totSlackcol, 0.0);
  if (error) goto QUIT;

  for (w = 0; w < nWorkers; ++w)
  {
    cind[w] = diffShiftscol(w);
    cval[w] = 1.0;
  }
  error = GRBaddqpterms(model, nWorkers, cind, cind, cval);
  if (error) goto QUIT;

  /* Optimize */
  error = solveAndPrint(model, nShifts, nWorkers, Workers, &status);
  if (error) goto QUIT;
  if (status != GRB_OPTIMAL) goto QUIT;


QUIT:

  /* Error reporting */

  if (error)
  {
    printf("ERROR: %s\n", GRBgeterrormsg(env));
    exit(1);
```

```
  }

  /* Free data */

  free(cbeg);
  free(cind);
  free(cval);
  free(sense);

  /* Free model */

  GRBfreemodel(model);

  /* Free environment */

  GRBfreeenv(env);

  return 0;
}


int solveAndPrint(GRBmodel* model,
                  int nShifts, int nWorkers, char** Workers,
                  int* status)
{
  int error, w;
  double val;

  error = GRBoptimize(model);
  if (error) return error;

  error = GRBgetintattr(model, "Status", status);
  if (error) return error;

  if ((*status == GRB_INF_OR_UNBD) || (*status == GRB_INFEASIBLE) ||
      (*status == GRB_UNBOUNDED))
  {
    printf("The model cannot be solved "
           "because it is infeasible or unbounded\n");
    return 0;
  }
  if (*status != GRB_OPTIMAL)
  {
    printf("Optimization was stopped with status %i\n", *status);
    return 0;
```

```
  }

  /* Print total slack and the number of shifts worked for each worker */
  error = GRBgetdblattrelement(model, "X", totSlackcol, &val);
  if (error) return error;

  printf("\nTotal slack required: %f\n", val);
  for (w = 0; w < nWorkers; ++w)
  {
    error = GRBgetdblattrelement(model, "X", totShiftscol(w), &val);
    if (error) return error;
    printf("%s worked %f shifts\n", Workers[w], val);
  }
  printf("\n");
  return 0;
}
```

workforce5_c.c

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* Assign workers to shifts; each worker may or may not be available on a
   particular day. We use multi-objective optimization to solve the model.
   The highest-priority objective minimizes the sum of the slacks
   (i.e., the total number of uncovered shifts). The secondary objective
   minimizes the difference between the maximum and minimum number of
   shifts worked among all workers.  The second optimization is allowed
   to degrade the first objective by up to the smaller value of 10% and 2 */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "gurobi_c.h"

int solveAndPrint(GRBmodel* model,
                  int nShifts, int nWorkers, char** Workers,
                  int* status);


#define xcol(w,s)         nShifts*w+s
#define slackcol(s)       nShifts*nWorkers+s
#define totSlackcol       nShifts*(nWorkers+1)
#define totShiftscol(w)   nShifts*(nWorkers+1)+1+w
#define minShiftcol       (nShifts+1)*(nWorkers+1)
#define maxShiftcol       (nShifts+1)*(nWorkers+1)+1
#define MAXSTR     128


int
main(int   argc,
     char *argv[])
{
  GRBenv   *env = NULL;
  GRBenv   *menv = NULL;
  GRBmodel *model = NULL;
  int       error = 0, status;
  int       s, w, col;
  int      *cbeg = NULL;
  int      *cind = NULL;
  int       idx;
  double   *cval = NULL;
  char     *sense = NULL;
```

```
char      vname[MAXSTR], cname[MAXSTR];

/* Sample data */
const int nShifts = 14;
const int nWorkers = 8;

/* Sets of days and workers */
char* Shifts[] =
  { "Mon1", "Tue2", "Wed3", "Thu4", "Fri5", "Sat6",
    "Sun7", "Mon8", "Tue9", "Wed10", "Thu11", "Fri12", "Sat13",
    "Sun14" };
char* Workers[] =
  { "Amy", "Bob", "Cathy", "Dan", "Ed", "Fred", "Gu", "Tobi" };

/* Number of workers required for each shift */
double shiftRequirements[] =
  { 3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5 };

/* Worker availability: 0 if the worker is unavailable for a shift */
double availability[][14] =
  { { 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1 },
    { 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0 },
    { 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
    { 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1 },
    { 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1 },
    { 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1 },
    { 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1 },
    { 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 } };

/* Create environment */
error = GRBloadenv(&env, "workforce5.log");
if (error) goto QUIT;

/* Create initial model */
error = GRBnewmodel(env, &model, "workforce5",
                    (nShifts + 1) * (nWorkers + 1) + 2,
                    NULL, NULL, NULL, NULL, NULL);
if (error) goto QUIT;

/* get model environment */
menv = GRBgetenv(model);
if (!menv) {
  fprintf(stderr, "Error: could not get model environment\n");
  goto QUIT;
}
```

```
/* Initialize assignment decision variables:
   x[w][s] == 1 if worker w is assigned to shift s.
   This is no longer a pure assignment model, so we must
   use binary variables. */
for (w = 0; w < nWorkers; ++w)
{
  for (s = 0; s < nShifts; ++s)
  {
    col = xcol(w, s);
    sprintf(vname, "%s.%s", Workers[w], Shifts[s]);
    error = GRBsetcharattrelement(model, "VType", col, GRB_BINARY);
    if (error) goto QUIT;
    error = GRBsetdblattrelement(model, "UB", col, availability[w][s]);
    if (error) goto QUIT;
    error = GRBsetstrattrelement(model, "VarName", col, vname);
    if (error) goto QUIT;
  }
}

/* Initialize slack decision variables */
for (s = 0; s < nShifts; ++s)
{
  sprintf(vname, "%sSlack", Shifts[s]);
  error = GRBsetstrattrelement(model, "VarName", slackcol(s), vname);
  if (error) goto QUIT;
}


/* Initialize total slack decision variable */
error = GRBsetstrattrelement(model, "VarName", totSlackcol, "totSlack");
if (error) goto QUIT;

/* Initialize variables to count the total shifts worked by each worker */
for (w = 0; w < nWorkers; ++w)
{
  sprintf(vname, "%sTotShifts", Workers[w]);
  error = GRBsetstrattrelement(model, "VarName", totShiftscol(w), vname);
  if (error) goto QUIT;
}

/* Initialize max and min #shifts variables */
sprintf(vname, "minShifts");
error = GRBsetstrattrelement(model, "VarName", minShiftcol, vname);
sprintf(vname, "maxShifts");
```

```
error = GRBsetstrattrelement(model, "VarName", maxShiftcol, vname);

/* Make space for constraint data */
cbeg = malloc(sizeof(int) * nShifts);
if (!cbeg) goto QUIT;
cind = malloc(sizeof(int) * nShifts * (nWorkers + 1));
if (!cind) goto QUIT;
cval = malloc(sizeof(double) * nShifts * (nWorkers + 1));
if (!cval) goto QUIT;
sense = malloc(sizeof(char) * (nShifts + nWorkers));
if (!sense) goto QUIT;

/* Constraint: assign exactly shiftRequirements[s] workers
   to each shift s, plus the slack */
idx = 0;
for (s = 0; s < nShifts; ++s)
{
  cbeg[s] = idx;
  sense[s] = GRB_EQUAL;
  for (w = 0; w < nWorkers; ++w)
  {
    cind[idx] = xcol(w, s);
    cval[idx++] = 1.0;
  }
  cind[idx] = slackcol(s);
  cval[idx++] = 1.0;
}
error = GRBaddconstrs(model, nShifts, idx, cbeg, cind, cval, sense,
                      shiftRequirements, Shifts);
if (error) goto QUIT;

/* Constraint: set totSlack column equal to the total slack */
idx = 0;
for (s = 0; s < nShifts; ++s)
{
  cind[idx] = slackcol(s);
  cval[idx++] = 1.0;
}
cind[idx] = totSlackcol;
cval[idx++] = -1.0;
error = GRBaddconstr(model, idx, cind, cval, GRB_EQUAL,
                     0.0, "totSlack");
if (error) goto QUIT;

/* Constraint: compute the total number of shifts for each worker */
```

```
for (w = 0; w < nWorkers; ++w)
{
  idx = 0;
  for (s = 0; s < nShifts; ++s)
  {
    cind[idx] = xcol(w,s);
    cval[idx++] = 1.0;
  }
  sprintf(cname, "totShifts%s", Workers[w]);
  cind[idx] = totShiftscol(w);
  cval[idx++] = -1.0;
  error = GRBaddconstr(model, idx, cind, cval, GRB_EQUAL, 0.0, cname);
  if (error) goto QUIT;
}

/* Constraint: set minShift/maxShift variable to less <=/>= to the
 * number of shifts among all workers */
for (w = 0; w < nWorkers; w++) {
  cind[w] = totShiftscol(w);
}
error = GRBaddgenconstrMin(model, NULL, minShiftcol, nWorkers, cind, GRB_INFINITY);
if (error) goto QUIT;
error = GRBaddgenconstrMax(model, NULL, maxShiftcol, nWorkers, cind, -GRB_INFINITY);
if (error) goto QUIT;

/* Set global sense for ALL objectives */
error = GRBsetintattr(model, GRB_INT_ATTR_MODELSENSE, GRB_MINIMIZE);
if (error) goto QUIT;

/* Set primary objective */
cind[0] = totSlackcol;
cval[0] = 1.0;
error = GRBsetobjectiven(model, 0, 2, 1.0, 2.0, 0.10, "TotalSlack",
                         0.0, 1, cind, cval);
if (error) goto QUIT;

/* Set secondary objective */
cind[0] = maxShiftcol;
cval[0] = 1.0;
cind[1] = minShiftcol;
cval[1] = -1.0;
error = GRBsetobjectiven(model, 1, 1, 1.0, 0, 0, "Fairness",
                         0.0, 2, cind, cval);
if (error) goto QUIT;
```

```
  /* Save problem */
  error = GRBwrite(model, "workforce5.lp");
  if (error) goto QUIT;
  error = GRBwrite(model, "workforce5.mps");
  if (error) goto QUIT;

  /* Optimize */
  error = solveAndPrint(model, nShifts, nWorkers, Workers, &status);
  if (error) goto QUIT;
  if (status != GRB_OPTIMAL) goto QUIT;

QUIT:

  /* Error reporting */

  if (error)
  {
    printf("ERROR: %s\n", GRBgeterrormsg(env));
    exit(1);
  }

  /* Free data */

  free(cbeg);
  free(cind);
  free(cval);
  free(sense);

  /* Free model */

  GRBfreemodel(model);

  /* Free environment */

  GRBfreeenv(env);

  return 0;
}


int solveAndPrint(GRBmodel* model,
                  int nShifts, int nWorkers, char** Workers,
                  int* status)
{
  int error, w;
```

```
  double val;

  error = GRBoptimize(model);
  if (error) return error;

  error = GRBgetintattr(model, "Status", status);
  if (error) return error;

  if ((*status == GRB_INF_OR_UNBD) || (*status == GRB_INFEASIBLE) ||
      (*status == GRB_UNBOUNDED))
  {
    printf("The model cannot be solved "
           "because it is infeasible or unbounded\n");
    return 0;
  }
  if (*status != GRB_OPTIMAL)
  {
    printf("Optimization was stopped with status %i\n", *status);
    return 0;
  }

  /* Print total slack and the number of shifts worked for each worker */
  error = GRBgetdblattrelement(model, "X", totSlackcol, &val);
  if (error) return error;

  printf("\nTotal slack required: %f\n", val);
  for (w = 0; w < nWorkers; ++w)
  {
    error = GRBgetdblattrelement(model, "X", totShiftscol(w), &val);
    if (error) return error;
    printf("%s worked %f shifts\n", Workers[w], val);
  }
  printf("\n");
  return 0;
}
```

3.2 C++ Examples

This section includes source code for all of the Gurobi C++ examples. The same source code can be found in the `examples/c++` directory of the Gurobi distribution.

callback_c++.cpp

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/*
   This example reads a model from a file, sets up a callback that
   monitors optimization progress and implements a custom
   termination strategy, and outputs progress information to the
   screen and to a log file.

   The termination strategy implemented in this callback stops the
   optimization of a MIP model once at least one of the following two
   conditions have been satisfied:
     1) The optimality gap is less than 10%
     2) At least 10000 nodes have been explored, and an integer feasible
        solution has been found.
   Note that termination is normally handled through Gurobi parameters
   (MIPGap, NodeLimit, etc.).  You should only use a callback for
   termination if the available parameters don't capture your desired
   termination criterion.
*/

#include "gurobi_c++.h"
#include <fstream>
#include <cmath>
using namespace std;

class mycallback: public GRBCallback
{
  public:
    double lastiter;
    double lastnode;
    int numvars;
    GRBVar* vars;
    ofstream* logfile;
    mycallback(int xnumvars, GRBVar* xvars, ofstream* xlogfile) {
      lastiter = lastnode = -GRB_INFINITY;
      numvars = xnumvars;
      vars = xvars;
      logfile = xlogfile;
    }
```

```
protected:
  void callback () {
    try {
      if (where == GRB_CB_POLLING) {
        // Ignore polling callback
      } else if (where == GRB_CB_PRESOLVE) {
        // Presolve callback
        int cdels = getIntInfo(GRB_CB_PRE_COLDEL);
        int rdels = getIntInfo(GRB_CB_PRE_ROWDEL);
        if (cdels || rdels) {
          cout << cdels << " columns and " << rdels
               << " rows are removed" << endl;
        }
      } else if (where == GRB_CB_SIMPLEX) {
        // Simplex callback
        double itcnt = getDoubleInfo(GRB_CB_SPX_ITRCNT);
        if (itcnt - lastiter >= 100) {
          lastiter = itcnt;
          double obj = getDoubleInfo(GRB_CB_SPX_OBJVAL);
          int ispert = getIntInfo(GRB_CB_SPX_ISPERT);
          double pinf = getDoubleInfo(GRB_CB_SPX_PRIMINF);
          double dinf = getDoubleInfo(GRB_CB_SPX_DUALINF);
          char ch;
          if (ispert == 0)      ch = ' ';
          else if (ispert == 1) ch = 'S';
          else                  ch = 'P';
          cout << itcnt << " " << obj << ch << " "
               << pinf << " " << dinf << endl;
        }
      } else if (where == GRB_CB_MIP) {
        // General MIP callback
        double nodecnt = getDoubleInfo(GRB_CB_MIP_NODCNT);
        double objbst = getDoubleInfo(GRB_CB_MIP_OBJBST);
        double objbnd = getDoubleInfo(GRB_CB_MIP_OBJBND);
        int solcnt = getIntInfo(GRB_CB_MIP_SOLCNT);
        if (nodecnt - lastnode >= 100) {
          lastnode = nodecnt;
          int actnodes = (int) getDoubleInfo(GRB_CB_MIP_NODLFT);
          int itcnt = (int) getDoubleInfo(GRB_CB_MIP_ITRCNT);
          int cutcnt = getIntInfo(GRB_CB_MIP_CUTCNT);
          cout << nodecnt << " " << actnodes << " " << itcnt
               << " " << objbst << " " << objbnd << " "
               << solcnt << " " << cutcnt << endl;
        }
        if (fabs(objbst - objbnd) < 0.1 * (1.0 + fabs(objbst))) {
```

```
        cout << "Stop early - 10% gap achieved" << endl;
        abort();
      }
      if (nodecnt >= 10000 && solcnt) {
        cout << "Stop early - 10000 nodes explored" << endl;
        abort();
      }
    } else if (where == GRB_CB_MIPSOL) {
      // MIP solution callback
      int nodecnt = (int) getDoubleInfo(GRB_CB_MIPSOL_NODCNT);
      double obj = getDoubleInfo(GRB_CB_MIPSOL_OBJ);
      int solcnt = getIntInfo(GRB_CB_MIPSOL_SOLCNT);
      double* x = getSolution(vars, numvars);
      cout << "**** New solution at node " << nodecnt
           << ", obj " << obj << ", sol " << solcnt
           << ", x[0] = " << x[0] << " ****" << endl;
      delete[] x;
    } else if (where == GRB_CB_MIPNODE) {
      // MIP node callback
      cout << "**** New node ****" << endl;
      if (getIntInfo(GRB_CB_MIPNODE_STATUS) == GRB_OPTIMAL) {
        double* x = getNodeRel(vars, numvars);
        setSolution(vars, x, numvars);
        delete[] x;
      }
    } else if (where == GRB_CB_BARRIER) {
      // Barrier callback
      int itcnt = getIntInfo(GRB_CB_BARRIER_ITRCNT);
      double primobj = getDoubleInfo(GRB_CB_BARRIER_PRIMOBJ);
      double dualobj = getDoubleInfo(GRB_CB_BARRIER_DUALOBJ);
      double priminf = getDoubleInfo(GRB_CB_BARRIER_PRIMINF);
      double dualinf = getDoubleInfo(GRB_CB_BARRIER_DUALINF);
      double cmpl = getDoubleInfo(GRB_CB_BARRIER_COMPL);
      cout << itcnt << " " << primobj << " " << dualobj << " "
           << priminf << " " << dualinf << " " << cmpl << endl;
    } else if (where == GRB_CB_MESSAGE) {
      // Message callback
      string msg = getStringInfo(GRB_CB_MSG_STRING);
      *logfile << msg;
    }
  } catch (GRBException e) {
    cout << "Error number: " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
  } catch (...) {
    cout << "Error during callback" << endl;
```

```
      }
    }
};

int
main(int   argc,
     char *argv[])
{
  if (argc < 2) {
    cout << "Usage: callback_c++ filename" << endl;
    return 1;
  }

  // Open log file
  ofstream logfile("cb.log");
  if (!logfile.is_open()) {
    cout << "Cannot open cb.log for callback message" << endl;
    return 1;
  }

  GRBEnv *env = 0;
  GRBVar *vars = 0;

  try {
    // Create environment
    env = new GRBEnv();

    // Read model from file
    GRBModel model = GRBModel(*env, argv[1]);

    // Turn off display and heuristics
    model.set(GRB_IntParam_OutputFlag, 0);
    model.set(GRB_DoubleParam_Heuristics, 0.0);

    // Create a callback object and associate it with the model
    int numvars = model.get(GRB_IntAttr_NumVars);
    vars = model.getVars();
    mycallback cb = mycallback(numvars, vars, &logfile);

    model.setCallback(&cb);

    // Solve model and capture solution information
    model.optimize();

    cout << endl << "Optimization complete" << endl;
```

```
    if (model.get(GRB_IntAttr_SolCount) == 0) {
      cout << "No solution found, optimization status = "
           << model.get(GRB_IntAttr_Status) << endl;
    } else {
      cout << "Solution found, objective = "
           << model.get(GRB_DoubleAttr_ObjVal) << endl;
      for (int j = 0; j < numvars; j++) {
        GRBVar v = vars[j];
        double x = v.get(GRB_DoubleAttr_X);
        if (x != 0.0) {
          cout << v.get(GRB_StringAttr_VarName) << " " << x << endl;
        }
      }
    }

  } catch (GRBException e) {
    cout << "Error number: " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
  } catch (...) {
    cout << "Error during optimization" << endl;
  }

  // Close log file
  logfile.close();

  delete[] vars;
  delete env;

  return 0;
}
```

dense_c++.cpp

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* This example formulates and solves the following simple QP model:

     minimize    x + y + x^2 + x*y + y^2 + y*z + z^2
     subject to  x + 2 y + 3 z >= 4
                 x +   y       >= 1

   The example illustrates the use of dense matrices to store A and Q
   (and dense vectors for the other relevant data).  We don't recommend
   that you use dense matrices, but this example may be helpful if you
   already have your data in this format.
*/

#include "gurobi_c++.h"
using namespace std;

static bool
dense_optimize(GRBEnv* env,
               int     rows,
               int     cols,
               double* c,     /* linear portion of objective function */
               double* Q,     /* quadratic portion of objective function */
               double* A,     /* constraint matrix */
               char*   sense, /* constraint senses */
               double* rhs,   /* RHS vector */
               double* lb,    /* variable lower bounds */
               double* ub,    /* variable upper bounds */
               char*   vtype, /* variable types (continuous, binary, etc.) */
               double* solution,
               double* objvalP)
{
  GRBModel model = GRBModel(*env);
  int i, j;
  bool success = false;

  /* Add variables to the model */

  GRBVar* vars = model.addVars(lb, ub, NULL, vtype, NULL, cols);

  /* Populate A matrix */

  for (i = 0; i < rows; i++) {
    GRBLinExpr lhs = 0;
```

```
    for (j = 0; j < cols; j++)
      if (A[i*cols+j] != 0)
        lhs += A[i*cols+j]*vars[j];
    model.addConstr(lhs, sense[i], rhs[i]);
  }

  GRBQuadExpr obj = 0;

  for (j = 0; j < cols; j++)
    obj += c[j]*vars[j];
  for (i = 0; i < cols; i++)
    for (j = 0; j < cols; j++)
      if (Q[i*cols+j] != 0)
        obj += Q[i*cols+j]*vars[i]*vars[j];

  model.setObjective(obj);

  model.optimize();

  model.write("dense.lp");

  if (model.get(GRB_IntAttr_Status) == GRB_OPTIMAL) {
    *objvalP = model.get(GRB_DoubleAttr_ObjVal);
    for (i = 0; i < cols; i++)
      solution[i] = vars[i].get(GRB_DoubleAttr_X);
    success = true;
  }

  delete[] vars;

  return success;
}

int
main(int   argc,
     char *argv[])
{
  GRBEnv* env = 0;
  try {
    env = new GRBEnv();
    double c[] = {1, 1, 0};
    double  Q[3][3] = {{1, 1, 0}, {0, 1, 1}, {0, 0, 1}};
    double  A[2][3] = {{1, 2, 3}, {1, 1, 0}};
    char    sense[] = {'>', '>'};
    double  rhs[]   = {4, 1};
```

```
    double  lb[]    = {0, 0, 0};
    bool    success;
    double  objval, sol[3];

    success = dense_optimize(env, 2, 3, c, &Q[0][0], &A[0][0], sense, rhs,
                             lb, NULL, NULL, sol, &objval);

    cout << "x: " << sol[0] << " y: " << sol[1] << " z: " << sol[2] << endl;

  } catch(GRBException e) {
    cout << "Error code = " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
  } catch(...) {
    cout << "Exception during optimization" << endl;
  }

  delete env;
  return 0;
}
```

diet_c++.cpp

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* Solve the classic diet model, showing how to add constraints
   to an existing model. */

#include "gurobi_c++.h"
using namespace std;

void printSolution(GRBModel& model, int nCategories, int nFoods,
                   GRBVar* buy, GRBVar* nutrition) throw(GRBException);

int
main(int argc,
     char *argv[])
{
  GRBEnv* env = 0;
  GRBVar* nutrition = 0;
  GRBVar* buy = 0;
  try
  {

    // Nutrition guidelines, based on
    // USDA Dietary Guidelines for Americans, 2005
    // http://www.health.gov/DietaryGuidelines/dga2005/
    const int nCategories = 4;
    string Categories[] =
      { "calories", "protein", "fat", "sodium" };
    double minNutrition[] = { 1800, 91, 0, 0 };
    double maxNutrition[] = { 2200, GRB_INFINITY, 65, 1779 };

    // Set of foods
    const int nFoods = 9;
    string Foods[] =
      { "hamburger", "chicken", "hot dog", "fries",
        "macaroni", "pizza", "salad", "milk", "ice cream" };
    double cost[] =
      { 2.49, 2.89, 1.50, 1.89, 2.09, 1.99, 2.49, 0.89, 1.59 };

    // Nutrition values for the foods
    double nutritionValues[][nCategories] = {
                      { 410, 24, 26, 730 },    // hamburger
                      { 420, 32, 10, 1190 },   // chicken
                      { 560, 20, 32, 1800 },   // hot dog
                      { 380, 4, 19, 270 },     // fries
```

```
                  { 320, 12, 10, 930 },    // macaroni
                  { 320, 15, 12, 820 },    // pizza
                  { 320, 31, 12, 1230 },   // salad
                  { 100, 8, 2.5, 125 },    // milk
                  { 330, 8, 10, 180 }      // ice cream
                };

// Model
env = new GRBEnv();
GRBModel model = GRBModel(*env);
model.set(GRB_StringAttr_ModelName, "diet");

// Create decision variables for the nutrition information,
// which we limit via bounds
nutrition = model.addVars(minNutrition, maxNutrition, 0, 0,
                          Categories, nCategories);

// Create decision variables for the foods to buy
buy = model.addVars(0, 0, cost, 0, Foods, nFoods);

// The objective is to minimize the costs
model.set(GRB_IntAttr_ModelSense, GRB_MINIMIZE);

// Nutrition constraints
for (int i = 0; i < nCategories; ++i)
{
  GRBLinExpr ntot = 0;
  for (int j = 0; j < nFoods; ++j)
  {
    ntot += nutritionValues[j][i] * buy[j];
  }
  model.addConstr(ntot == nutrition[i], Categories[i]);
}

// Solve
model.optimize();
printSolution(model, nCategories, nFoods, buy, nutrition);

cout << "\nAdding constraint: at most 6 servings of dairy" << endl;
model.addConstr(buy[7] + buy[8] <= 6.0, "limit_dairy");

// Solve
model.optimize();
printSolution(model, nCategories, nFoods, buy, nutrition);
```

```
  }
  catch (GRBException e)
  {
    cout << "Error code = " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
  }
  catch (...)
  {
    cout << "Exception during optimization" << endl;
  }

  delete[] nutrition;
  delete[] buy;
  delete env;
  return 0;
}

void printSolution(GRBModel& model, int nCategories, int nFoods,
                   GRBVar* buy, GRBVar* nutrition) throw(GRBException)
{
  if (model.get(GRB_IntAttr_Status) == GRB_OPTIMAL)
  {
    cout << "\nCost: " << model.get(GRB_DoubleAttr_ObjVal) << endl;
    cout << "\nBuy:" << endl;
    for (int j = 0; j < nFoods; ++j)
    {
      if (buy[j].get(GRB_DoubleAttr_X) > 0.0001)
      {
        cout << buy[j].get(GRB_StringAttr_VarName) << " " <<
        buy[j].get(GRB_DoubleAttr_X) << endl;
      }
    }
    cout << "\nNutrition:" << endl;
    for (int i = 0; i < nCategories; ++i)
    {
      cout << nutrition[i].get(GRB_StringAttr_VarName) << " " <<
      nutrition[i].get(GRB_DoubleAttr_X) << endl;
    }
  }
  else
  {
    cout << "No solution" << endl;
  }
}
```

facility_c++.cpp

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* Facility location: a company currently ships its product from 5 plants
   to 4 warehouses. It is considering closing some plants to reduce
   costs. What plant(s) should the company close, in order to minimize
   transportation and fixed costs?

   Based on an example from Frontline Systems:
   http://www.solver.com/disfacility.htm
   Used with permission.
 */

#include "gurobi_c++.h"
#include <sstream>
using namespace std;

int
main(int argc,
     char *argv[])
{
  GRBEnv* env = 0;
  GRBVar* open = 0;
  GRBVar** transport = 0;
  int transportCt = 0;
  try
  {

    // Number of plants and warehouses
    const int nPlants = 5;
    const int nWarehouses = 4;

    // Warehouse demand in thousands of units
    double Demand[] = { 15, 18, 14, 20 };

    // Plant capacity in thousands of units
    double Capacity[] = { 20, 22, 17, 19, 18 };

    // Fixed costs for each plant
    double FixedCosts[] =
      { 12000, 15000, 17000, 13000, 16000 };

    // Transportation costs per thousand units
    double TransCosts[][nPlants] = {
                                    { 4000, 2000, 3000, 2500, 4500 },
```

```
                                  { 2500, 2600, 3400, 3000, 4000 },
                                  { 1200, 1800, 2600, 4100, 3000 },
                                  { 2200, 2600, 3100, 3700, 3200 }
                                };

// Model
env = new GRBEnv();
GRBModel model = GRBModel(*env);
model.set(GRB_StringAttr_ModelName, "facility");

// Plant open decision variables: open[p] == 1 if plant p is open.
open = model.addVars(nPlants, GRB_BINARY);

int p;
for (p = 0; p < nPlants; ++p)
{
  ostringstream vname;
  vname << "Open" << p;
  open[p].set(GRB_DoubleAttr_Obj, FixedCosts[p]);
  open[p].set(GRB_StringAttr_VarName, vname.str());
}

// Transportation decision variables: how much to transport from
// a plant p to a warehouse w
transport = new GRBVar* [nWarehouses];
int w;
for (w = 0; w < nWarehouses; ++w)
{
  transport[w] = model.addVars(nPlants);
  transportCt++;

  for (p = 0; p < nPlants; ++p)
  {
    ostringstream vname;
    vname << "Trans" << p << "." << w;
    transport[w][p].set(GRB_DoubleAttr_Obj, TransCosts[w][p]);
    transport[w][p].set(GRB_StringAttr_VarName, vname.str());
  }
}

// The objective is to minimize the total fixed and variable costs
model.set(GRB_IntAttr_ModelSense, GRB_MINIMIZE);

// Production constraints
// Note that the right-hand limit sets the production to zero if
```

```
// the plant is closed
for (p = 0; p < nPlants; ++p)
{
  GRBLinExpr ptot = 0;
  for (w = 0; w < nWarehouses; ++w)
  {
    ptot += transport[w][p];
  }
  ostringstream cname;
  cname << "Capacity" << p;
  model.addConstr(ptot <= Capacity[p] * open[p], cname.str());
}

// Demand constraints
for (w = 0; w < nWarehouses; ++w)
{
  GRBLinExpr dtot = 0;
  for (p = 0; p < nPlants; ++p)
  {
    dtot += transport[w][p];
  }
  ostringstream cname;
  cname << "Demand" << w;
  model.addConstr(dtot == Demand[w], cname.str());
}

// Guess at the starting point: close the plant with the highest
// fixed costs; open all others

// First, open all plants
for (p = 0; p < nPlants; ++p)
{
  open[p].set(GRB_DoubleAttr_Start, 1.0);
}

// Now close the plant with the highest fixed cost
cout << "Initial guess:" << endl;
double maxFixed = -GRB_INFINITY;
for (p = 0; p < nPlants; ++p)
{
  if (FixedCosts[p] > maxFixed)
  {
    maxFixed = FixedCosts[p];
  }
}
```

```
  for (p = 0; p < nPlants; ++p)
  {
    if (FixedCosts[p] == maxFixed)
    {
      open[p].set(GRB_DoubleAttr_Start, 0.0);
      cout << "Closing plant " << p << endl << endl;
      break;
    }
  }

  // Use barrier to solve root relaxation
  model.set(GRB_IntParam_Method, GRB_METHOD_BARRIER);

  // Solve
  model.optimize();

  // Print solution
  cout << "\nTOTAL COSTS: " << model.get(GRB_DoubleAttr_ObjVal) << endl;
  cout << "SOLUTION:" << endl;
  for (p = 0; p < nPlants; ++p)
  {
    if (open[p].get(GRB_DoubleAttr_X) > 0.99)
    {
      cout << "Plant " << p << " open:" << endl;
      for (w = 0; w < nWarehouses; ++w)
      {
        if (transport[w][p].get(GRB_DoubleAttr_X) > 0.0001)
        {
          cout << "  Transport " <<
          transport[w][p].get(GRB_DoubleAttr_X) <<
          " units to warehouse " << w << endl;
        }
      }
    }
    else
    {
      cout << "Plant " << p << " closed!" << endl;
    }
  }

}
catch (GRBException e)
{
  cout << "Error code = " << e.getErrorCode() << endl;
  cout << e.getMessage() << endl;
```

```
  }
  catch (...)
  {
    cout << "Exception during optimization" << endl;
  }

  delete[] open;
  for (int i = 0; i < transportCt; ++i) {
    delete[] transport[i];
  }
  delete[] transport;
  delete env;
  return 0;
}
```

feasopt_c++.cpp

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* This example reads a MIP model from a file, adds artificial
   variables to each constraint, and then minimizes the sum of the
   artificial variables.  A solution with objective zero corresponds
   to a feasible solution to the input model.
   We can also use FeasRelax feature to do it. In this example, we
   use minrelax=1, i.e. optimizing the returned model finds a solution
   that minimizes the original objective, but only from among those
   solutions that minimize the sum of the artificial variables. */

#include "gurobi_c++.h"
using namespace std;

int
main(int argc,
     char *argv[])
{
  if (argc < 2)
  {
    cout << "Usage: feasopt_c++ filename" << endl;
    return 1;
  }

  GRBEnv* env = 0;
  GRBConstr* c = 0;
  try
  {
    env = new GRBEnv();
    GRBModel feasmodel = GRBModel(*env, argv[1]);

    // Create a copy to use FeasRelax feature later */
    GRBModel feasmodel1 = GRBModel(feasmodel);

    // clear objective
    feasmodel.setObjective(GRBLinExpr(0.0));

    // add slack variables
    c = feasmodel.getConstrs();
    for (int i = 0; i < feasmodel.get(GRB_IntAttr_NumConstrs); ++i)
    {
      char sense = c[i].get(GRB_CharAttr_Sense);
      if (sense != '>')
      {
```

```
        double coef = -1.0;
        feasmodel.addVar(0.0, GRB_INFINITY, 1.0, GRB_CONTINUOUS, 1,
                         &c[i], &coef, "ArtN_" +
                         c[i].get(GRB_StringAttr_ConstrName));
      }
      if (sense != '<')
      {
        double coef = 1.0;
        feasmodel.addVar(0.0, GRB_INFINITY, 1.0, GRB_CONTINUOUS, 1,
                         &c[i], &coef, "ArtP_" +
                         c[i].get(GRB_StringAttr_ConstrName));
      }
    }

    // optimize modified model
    feasmodel.optimize();
    feasmodel.write("feasopt.lp");

    // use FeasRelax feature */
    feasmodel1.feasRelax(GRB_FEASRELAX_LINEAR, true, false, true);
    feasmodel1.write("feasopt1.lp");
    feasmodel1.optimize();
  }
  catch (GRBException e)
  {
    cout << "Error code = " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
  }
  catch (...)
  {
    cout << "Error during optimization" << endl;
  }

  delete[] c;
  delete env;
  return 0;
}
```

fixanddive_c++.cpp

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* Implement a simple MIP heuristic.  Relax the model,
   sort variables based on fractionality, and fix the 25% of
   the fractional variables that are closest to integer variables.
   Repeat until either the relaxation is integer feasible or
   linearly infeasible. */

#include "gurobi_c++.h"
#include <algorithm>
#include <cmath>
#include <deque>
using namespace std;

bool vcomp(GRBVar*, GRBVar*) throw(GRBException);

int
main(int argc,
     char *argv[])
{
  if (argc < 2)
  {
    cout << "Usage: fixanddive_c++ filename" << endl;
    return 1;
  }

  GRBEnv* env = 0;
  GRBVar* x = 0;
  try
  {
    // Read model
    env = new GRBEnv();
    GRBModel model = GRBModel(*env, argv[1]);

    // Collect integer variables and relax them
    // Note that we use GRBVar* to copy variables
    deque<GRBVar*> intvars;
    x = model.getVars();
    for (int j = 0; j < model.get(GRB_IntAttr_NumVars); ++j)
    {
      if (x[j].get(GRB_CharAttr_VType) != GRB_CONTINUOUS)
      {
        intvars.push_back(&x[j]);
        x[j].set(GRB_CharAttr_VType, GRB_CONTINUOUS);
```

```
  }
}

model.set(GRB_IntParam_OutputFlag, 0);
model.optimize();

// Perform multiple iterations. In each iteration, identify the first
// quartile of integer variables that are closest to an integer value
// in the relaxation, fix them to the nearest integer, and repeat.

for (int iter = 0; iter < 1000; ++iter)
{

  // create a list of fractional variables, sorted in order of
  // increasing distance from the relaxation solution to the nearest
  // integer value

  deque<GRBVar*> fractional;
  for (size_t j = 0; j < intvars.size(); ++j)
  {
    double sol = fabs(intvars[j]->get(GRB_DoubleAttr_X));
    if (fabs(sol - floor(sol + 0.5)) > 1e-5)
    {
      fractional.push_back(intvars[j]);
    }
  }

  cout << "Iteration " << iter << ", obj " <<
  model.get(GRB_DoubleAttr_ObjVal) << ", fractional " <<
  fractional.size() << endl;

  if (fractional.size() == 0)
  {
    cout << "Found feasible solution - objective " <<
    model.get(GRB_DoubleAttr_ObjVal) << endl;
    break;
  }

  // Fix the first quartile to the nearest integer value
  sort(fractional.begin(), fractional.end(), vcomp);
  int nfix = fractional.size() / 4;
  nfix = (nfix > 1) ? nfix : 1;
  for (int i = 0; i < nfix; ++i)
  {
    GRBVar* v = fractional[i];
```

```
        double fixval = floor(v->get(GRB_DoubleAttr_X) + 0.5);
        v->set(GRB_DoubleAttr_LB, fixval);
        v->set(GRB_DoubleAttr_UB, fixval);
        cout << "  Fix " << v->get(GRB_StringAttr_VarName) << " to " <<
        fixval << " ( rel " << v->get(GRB_DoubleAttr_X) << " )" <<
        endl;
      }

      model.optimize();

      // Check optimization result

      if (model.get(GRB_IntAttr_Status) != GRB_OPTIMAL)
      {
        cout << "Relaxation is infeasible" << endl;
        break;
      }
    }

  }
  catch (GRBException e)
  {
    cout << "Error code = " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
  }
  catch (...)
  {
    cout << "Error during optimization" << endl;
  }

  delete[] x;
  delete env;
  return 0;
}


bool vcomp(GRBVar* v1,
           GRBVar* v2) throw(GRBException)
{
  double sol1 = fabs(v1->get(GRB_DoubleAttr_X));
  double sol2 = fabs(v2->get(GRB_DoubleAttr_X));
  double frac1 = fabs(sol1 - floor(sol1 + 0.5));
  double frac2 = fabs(sol2 - floor(sol2 + 0.5));
  return (frac1 < frac2);
}
```

genconstr_c++.cpp

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* In this example we show the use of general constraints for modeling
 * some common expressions. We use as an example a SAT-problem where we
 * want to see if it is possible to satisfy at least four (or all) clauses
 * of the logical for
 *
 * L = (x0 or ~x1 or x2)  and (x1 or ~x2 or x3)  and
 *     (x2 or ~x3 or x0)  and (x3 or ~x0 or x1)  and
 *     (~x0 or ~x1 or x2) and (~x1 or ~x2 or x3) and
 *     (~x2 or ~x3 or x0) and (~x3 or ~x0 or x1)
 *
 * We do this by introducing two variables for each literal (itself and its
 * negated value), a variable for each clause, and then two
 * variables for indicating if we can satisfy four, and another to identify
 * the minimum of the clauses (so if it one, we can satisfy all clauses)
 * and put these two variables in the objective.
 * i.e. the Objective function will be
 *
 * maximize Obj0 + Obj1
 *
 *  Obj0 = MIN(Clause1, ... , Clause8)
 *  Obj1 = 1 -> Clause1 + ... + Clause8 >= 4
 *
 * thus, the objective value will be two if and only if we can satisfy all
 * clauses; one if and only if at least four clauses can be satisfied, and
 * zero otherwise.
 */


#include "gurobi_c++.h"
#include <sstream>
#include <iomanip>
using namespace std;

#define n         4
#define NLITERALS 4  // same as n
#define NCLAUSES  8
#define NOBJ      2

int
main(void)
{
  GRBEnv *env = 0;
```

```
try{
  // Example data
  //   e.g. {0, n+1, 2} means clause (x0 or ~x1 or x2)
  const int Clauses[][3] = {{  0, n+1, 2}, {  1, n+2, 3},
                            {  2, n+3, 0}, {  3, n+0, 1},
                            {n+0, n+1, 2}, {n+1, n+2, 3},
                            {n+2, n+3, 0}, {n+3, n+0, 1}};

  int i, status, nSolutions;

  // Create environment
  env = new GRBEnv("genconstr_c++.log");

  // Create initial model
  GRBModel model = GRBModel(*env);
  model.set(GRB_StringAttr_ModelName, "genconstr_c++");

  // Initialize decision variables and objective

  GRBVar Lit[NLITERALS];
  GRBVar NotLit[NLITERALS];
  for (i = 0; i < NLITERALS; i++) {
    ostringstream vname;
    vname << "X" << i;
    Lit[i]    = model.addVar(0.0, 1.0, 0.0, GRB_BINARY, vname.str());

    vname.str("");
    vname << "notX" << i;
    NotLit[i] = model.addVar(0.0, 1.0, 0.0, GRB_BINARY, vname.str());
  }

  GRBVar Cla[NCLAUSES];
  for (i = 0; i < NCLAUSES; i++) {
    ostringstream vname;
    vname << "Clause" << i;
    Cla[i] = model.addVar(0.0, 1.0, 0.0, GRB_BINARY, vname.str());
  }

  GRBVar Obj[NOBJ];
  for (i = 0; i < NOBJ; i++) {
    ostringstream vname;
    vname << "Obj" << i;
    Obj[i] = model.addVar(0.0, 1.0, 1.0, GRB_BINARY, vname.str());
  }
```

```
// Link Xi and notXi
GRBLinExpr lhs;
for (i = 0; i < NLITERALS; i++) {
  ostringstream cname;
  cname << "CNSTR_X" << i;
  lhs = 0;
  lhs += Lit[i];
  lhs += NotLit[i];
  model.addConstr(lhs == 1.0, cname.str());
}

// Link clauses and literals
GRBVar clause[3];
for (i = 0; i < NCLAUSES; i++) {
  for (int j = 0; j < 3; j++) {
    if (Clauses[i][j] >= n) clause[j] = NotLit[Clauses[i][j]-n];
    else                    clause[j] = Lit[Clauses[i][j]];
  }
  ostringstream cname;
  cname << "CNSTR_Clause" << i;
  model.addGenConstrOr(Cla[i], clause, 3, cname.str());
}

// Link objs with clauses
model.addGenConstrMin(Obj[0], Cla, NCLAUSES,
                      GRB_INFINITY, "CNSTR_Obj0");
lhs = 0;
for (i = 0; i < NCLAUSES; i++) {
  lhs += Cla[i];
}
model.addGenConstrIndicator(Obj[1], 1, lhs >= 4.0, "CNSTR_Obj1");

// Set global objective sense
model.set(GRB_IntAttr_ModelSense, GRB_MAXIMIZE);

// Save problem
model.write("genconstr_c++.mps");
model.write("genconstr_c++.lp");

// Optimize
model.optimize();

// Status checking
status = model.get(GRB_IntAttr_Status);
```

```
    if (status == GRB_INF_OR_UNBD ||
        status == GRB_INFEASIBLE  ||
        status == GRB_UNBOUNDED     ) {
      cout << "The model cannot be solved " <<
              "because it is infeasible or unbounded" << endl;
      return 1;
    }
    if (status != GRB_OPTIMAL) {
      cout << "Optimization was stopped with status " << status << endl;
      return 1;
    }

    // Print result
    double objval = model.get(GRB_DoubleAttr_ObjVal);

    if (objval > 1.9)
      cout << "Logical expression is satisfiable" << endl;
    else if (objval > 0.9)
      cout << "At least four clauses can be satisfied" << endl;
    else
      cout << "Not even three clauses can be satisfied" << endl;

  } catch (GRBException e) {
    cout << "Error code = " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
  }
  catch (...) {
    cout << "Exception during optimization" << endl;
  }

  // Free environment
  delete env;

  return 0;
}
```

lp_c++.cpp

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* This example reads an LP model from a file and solves it.
   If the model is infeasible or unbounded, the example turns off
   presolve and solves the model again. If the model is infeasible,
   the example computes an Irreducible Inconsistent Subsystem (IIS),
   and writes it to a file */

#include "gurobi_c++.h"
using namespace std;

int
main(int   argc,
     char *argv[])
{
  if (argc < 2) {
    cout << "Usage: lp_c++ filename" << endl;
    return 1;
  }

  try {
    GRBEnv env = GRBEnv();
    GRBModel model = GRBModel(env, argv[1]);

    model.optimize();

    int optimstatus = model.get(GRB_IntAttr_Status);

    if (optimstatus == GRB_INF_OR_UNBD) {
      model.set(GRB_IntParam_Presolve, 0);
      model.optimize();
      optimstatus = model.get(GRB_IntAttr_Status);
    }

    if (optimstatus == GRB_OPTIMAL) {
      double objval = model.get(GRB_DoubleAttr_ObjVal);
      cout << "Optimal objective: " << objval << endl;
    } else if (optimstatus == GRB_INFEASIBLE) {
      cout << "Model is infeasible" << endl;

      // compute and write out IIS

      model.computeIIS();
      model.write("model.ilp");
```

```
    } else if (optimstatus == GRB_UNBOUNDED) {
      cout << "Model is unbounded" << endl;
    } else {
      cout << "Optimization was stopped with status = "
           << optimstatus << endl;
    }

  } catch(GRBException e) {
    cout << "Error code = " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
  } catch (...) {
    cout << "Error during optimization" << endl;
  }

  return 0;
}
```

lpmethod_c++.cpp

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* Solve a model with different values of the Method parameter;
   show which value gives the shortest solve time. */

#include "gurobi_c++.h"
using namespace std;

int
main(int   argc,
     char *argv[])
{

  if (argc < 2)
  {
    cout << "Usage: lpmethod_c++ filename" << endl;
    return 1;
  }

  try {
    // Read model
    GRBEnv env = GRBEnv();
    GRBModel m = GRBModel(env, argv[1]);

    // Solve the model with different values of Method
    int    bestMethod = -1;
    double bestTime = m.get(GRB_DoubleParam_TimeLimit);
    for (int i = 0; i <= 2; ++i) {
      m.reset();
      m.set(GRB_IntParam_Method, i);
      m.optimize();
      if (m.get(GRB_IntAttr_Status) == GRB_OPTIMAL) {
        bestTime = m.get(GRB_DoubleAttr_Runtime);
        bestMethod = i;
        // Reduce the TimeLimit parameter to save time
        // with other methods
        m.set(GRB_DoubleParam_TimeLimit, bestTime);
      }
    }

    // Report which method was fastest
    if (bestMethod == -1) {
      cout << "Unable to solve this model" << endl;
    } else {
```

```
      cout << "Solved in " << bestTime
        << " seconds with Method: " << bestMethod << endl;
    }
  } catch(GRBException e) {
    cout << "Error code = " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
  } catch(...) {
    cout << "Exception during optimization" << endl;
  }

  return 0;
}
```

lpmod_c++.cpp

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* This example reads an LP model from a file and solves it.
   If the model can be solved, then it finds the smallest positive variable,
   sets its upper bound to zero, and resolves the model two ways:
   first with an advanced start, then without an advanced start
   (i.e. 'from scratch'). */

#include "gurobi_c++.h"
using namespace std;

int
main(int argc,
     char *argv[])
{
  if (argc < 2)
  {
    cout << "Usage: lpmod_c++ filename" << endl;
    return 1;
  }

  GRBEnv* env = 0;
  GRBVar* v = 0;
  try
  {
    // Read model and determine whether it is an LP
    env = new GRBEnv();
    GRBModel model = GRBModel(*env, argv[1]);
    if (model.get(GRB_IntAttr_IsMIP) != 0)
    {
      cout << "The model is not a linear program" << endl;
      return 1;
    }

    model.optimize();

    int status = model.get(GRB_IntAttr_Status);

    if ((status == GRB_INF_OR_UNBD) || (status == GRB_INFEASIBLE) ||
        (status == GRB_UNBOUNDED))
    {
      cout << "The model cannot be solved because it is "
      << "infeasible or unbounded" << endl;
      return 1;
```

```
}

if (status != GRB_OPTIMAL)
{
  cout << "Optimization was stopped with status " << status << endl;
  return 0;
}

// Find the smallest variable value
double minVal = GRB_INFINITY;
int minVar = 0;
v = model.getVars();
for (int j = 0; j < model.get(GRB_IntAttr_NumVars); ++j)
{
  double sol = v[j].get(GRB_DoubleAttr_X);
  if ((sol > 0.0001) && (sol < minVal) &&
      (v[j].get(GRB_DoubleAttr_LB) == 0.0))
  {
    minVal = sol;
    minVar = j;
  }
}

cout << "\n*** Setting " << v[minVar].get(GRB_StringAttr_VarName)
<< " from " << minVal << " to zero ***" << endl << endl;
v[minVar].set(GRB_DoubleAttr_UB, 0.0);

// Solve from this starting point
model.optimize();

// Save iteration & time info
double warmCount = model.get(GRB_DoubleAttr_IterCount);
double warmTime = model.get(GRB_DoubleAttr_Runtime);

// Reset the model and resolve
cout << "\n*** Resetting and solving "
<< "without an advanced start ***\n" << endl;
model.reset();
model.optimize();

// Save iteration & time info
double coldCount = model.get(GRB_DoubleAttr_IterCount);
double coldTime = model.get(GRB_DoubleAttr_Runtime);

cout << "\n*** Warm start: " << warmCount << " iterations, " <<
```

```
    warmTime << " seconds" << endl;
    cout << "*** Cold start: " << coldCount << " iterations, " <<
    coldTime << " seconds" << endl;

  }
  catch (GRBException e)
  {
    cout << "Error code = " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
  }
  catch (...)
  {
    cout << "Error during optimization" << endl;
  }

  delete[] v;
  delete env;
  return 0;
}
```

mip1_c++.cpp

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* This example formulates and solves the following simple MIP model:

     maximize    x +   y + 2 z
     subject to  x + 2 y + 3 z <= 4
                 x +   y       >= 1
     x, y, z binary
*/

#include "gurobi_c++.h"
using namespace std;

int
main(int   argc,
     char *argv[])
{
  try {
    GRBEnv env = GRBEnv();

    GRBModel model = GRBModel(env);

    // Create variables

    GRBVar x = model.addVar(0.0, 1.0, 0.0, GRB_BINARY, "x");
    GRBVar y = model.addVar(0.0, 1.0, 0.0, GRB_BINARY, "y");
    GRBVar z = model.addVar(0.0, 1.0, 0.0, GRB_BINARY, "z");

    // Set objective: maximize x + y + 2 z

    model.setObjective(x + y + 2 * z, GRB_MAXIMIZE);

    // Add constraint: x + 2 y + 3 z <= 4

    model.addConstr(x + 2 * y + 3 * z <= 4, "c0");

    // Add constraint: x + y >= 1

    model.addConstr(x + y >= 1, "c1");

    // Optimize model

    model.optimize();
```

```
    cout << x.get(GRB_StringAttr_VarName) << " "
         << x.get(GRB_DoubleAttr_X) << endl;
    cout << y.get(GRB_StringAttr_VarName) << " "
         << y.get(GRB_DoubleAttr_X) << endl;
    cout << z.get(GRB_StringAttr_VarName) << " "
         << z.get(GRB_DoubleAttr_X) << endl;

    cout << "Obj: " << model.get(GRB_DoubleAttr_ObjVal) << endl;

  } catch(GRBException e) {
    cout << "Error code = " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
  } catch(...) {
    cout << "Exception during optimization" << endl;
  }

  return 0;
}
```

mip2_c++.cpp

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* This example reads a MIP model from a file, solves it and
   prints the objective values from all feasible solutions
   generated while solving the MIP. Then it creates the fixed
   model and solves that model. */

#include "gurobi_c++.h"
#include <cmath>
using namespace std;

int
main(int   argc,
     char *argv[])
{
  if (argc < 2) {
    cout << "Usage: mip2_c++ filename" << endl;
    return 1;
  }

  GRBEnv *env = 0;
  GRBVar *vars = 0, *fvars = 0;
  try {
    env = new GRBEnv();
    GRBModel model = GRBModel(*env, argv[1]);

    if (model.get(GRB_IntAttr_IsMIP) == 0) {
      throw GRBException("Model is not a MIP");
    }

    model.optimize();

    int optimstatus = model.get(GRB_IntAttr_Status);

    cout << "Optimization complete" << endl;
    double objval = 0;
    if (optimstatus == GRB_OPTIMAL) {
      objval = model.get(GRB_DoubleAttr_ObjVal);
      cout << "Optimal objective: " << objval << endl;
    } else if (optimstatus == GRB_INF_OR_UNBD) {
      cout << "Model is infeasible or unbounded" << endl;
      return 0;
    } else if (optimstatus == GRB_INFEASIBLE) {
      cout << "Model is infeasible" << endl;
```

```
  return 0;
} else if (optimstatus == GRB_UNBOUNDED) {
  cout << "Model is unbounded" << endl;
  return 0;
} else {
  cout << "Optimization was stopped with status = "
       << optimstatus << endl;
  return 0;
}

/* Iterate over the solutions and compute the objectives */

int numvars = model.get(GRB_IntAttr_NumVars);
vars = model.getVars();
model.set(GRB_IntParam_OutputFlag, 0);

cout << endl;
for ( int k = 0; k < model.get(GRB_IntAttr_SolCount); ++k ) {
  model.set(GRB_IntParam_SolutionNumber, k);
  double objn = 0.0;

  for (int j = 0; j < numvars; j++) {
    GRBVar v = vars[j];
    objn += v.get(GRB_DoubleAttr_Obj) * v.get(GRB_DoubleAttr_Xn);
  }

  cout << "Solution " << k << " has objective: " << objn << endl;
}
cout << endl;
model.set(GRB_IntParam_OutputFlag, 1);

/* Create a fixed model, turn off presolve and solve */

GRBModel fixed = model.fixedModel();

fixed.set(GRB_IntParam_Presolve, 0);

fixed.optimize();

int foptimstatus = fixed.get(GRB_IntAttr_Status);

if (foptimstatus != GRB_OPTIMAL) {
  cerr << "Error: fixed model isn't optimal" << endl;
  return 0;
}
```

```
    double fobjval = fixed.get(GRB_DoubleAttr_ObjVal);

    if (fabs(fobjval - objval) > 1.0e-6 * (1.0 + fabs(objval))) {
      cerr << "Error: objective values are different" << endl;
      return 0;
    }

    /* Print values of nonzero variables */
    fvars = fixed.getVars();
    for (int j = 0; j < numvars; j++) {
      GRBVar v = fvars[j];
      if (v.get(GRB_DoubleAttr_X) != 0.0) {
        cout << v.get(GRB_StringAttr_VarName) << " "
             << v.get(GRB_DoubleAttr_X) << endl;
      }
    }

  } catch(GRBException e) {
    cout << "Error code = " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
  } catch (...) {
    cout << "Error during optimization" << endl;
  }

  delete[] fvars;
  delete[] vars;
  delete env;
  return 0;
}
```

multiobj_c++.cpp

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* Want to cover three different sets but subject to a common budget of
 * elements allowed to be used. However, the sets have different priorities to
 * be covered; and we tackle this by using multi-objective optimization. */

#include "gurobi_c++.h"
#include <sstream>
#include <iomanip>
using namespace std;

int
main(void)
{
  GRBEnv *env  = 0;
  GRBVar *Elem = 0;
  int e, i, status, nSolutions;

  try{
    // Sample data
    const int groundSetSize = 20;
    const int nSubsets      = 4;
    const int Budget        = 12;
    double Set[][20] =
    { { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
      { 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1 },
      { 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0 },
      { 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0 } };
    int    SetObjPriority[] = {3, 2, 2, 1};
    double SetObjWeight[]   = {1.0, 0.25, 1.25, 1.0};

    // Create environment
    env = new GRBEnv("multiobj_c++.log");

    // Create initial model
    GRBModel model = GRBModel(*env);
    model.set(GRB_StringAttr_ModelName, "multiobj_c++");

    // Initialize decision variables for ground set:
    // x[e] == 1 if element e is chosen for the covering.
    Elem = model.addVars(groundSetSize, GRB_BINARY);
    for (e = 0; e < groundSetSize; e++) {
      ostringstream vname;
      vname << "El" << e;
```

```
  Elem[e].set(GRB_StringAttr_VarName, vname.str());
}

// Constraint: limit total number of elements to be picked to be at most
// Budget
GRBLinExpr lhs;
lhs = 0;
for (e = 0; e < groundSetSize; e++) {
  lhs += Elem[e];
}
model.addConstr(lhs <= Budget, "Budget");

// Set global sense for ALL objectives
model.set(GRB_IntAttr_ModelSense, GRB_MAXIMIZE);

// Limit how many solutions to collect
model.set(GRB_IntParam_PoolSolutions, 100);

// Set and configure i-th objective
for (i = 0; i < nSubsets; i++) {
  GRBLinExpr objn = 0;
  for (e = 0; e < groundSetSize; e++)
    objn += Set[i][e]*Elem[e];
  ostringstream vname;
  vname << "Set" << i;

  model.setObjectiveN(objn, i, SetObjPriority[i], SetObjWeight[i],
                      1.0 + i, 0.01, vname.str());
}

// Save problem
model.write("multiobj_c++.lp");

// Optimize
model.optimize();

// Status checking
status = model.get(GRB_IntAttr_Status);

if (status == GRB_INF_OR_UNBD ||
    status == GRB_INFEASIBLE  ||
    status == GRB_UNBOUNDED     ) {
  cout << "The model cannot be solved " <<
         "because it is infeasible or unbounded" << endl;
  return 1;
```

```
  }
  if (status != GRB_OPTIMAL) {
    cout << "Optimization was stopped with status " << status << endl;
    return 1;
  }

  // Print best selected set
  cout << "Selected elements in best solution:" << endl << "\t";
  for (e = 0; e < groundSetSize; e++) {
    if (Elem[e].get(GRB_DoubleAttr_X) < .9) continue;
    cout << " El" << e;
  }
  cout << endl;

  // Print number of solutions stored
  nSolutions = model.get(GRB_IntAttr_SolCount);
  cout << "Number of solutions found: " << nSolutions << endl;

  // Print objective values of solutions
  if (nSolutions > 10) nSolutions = 10;
  cout << "Objective values for first " << nSolutions;
  cout << " solutions:" << endl;
  for (i = 0; i < nSubsets; i++) {
    model.set(GRB_IntParam_ObjNumber, i);

    cout << "\tSet" << i;
    for (e = 0; e < nSolutions; e++) {
      cout << " ";
      model.set(GRB_IntParam_SolutionNumber, e);
      double val = model.get(GRB_DoubleAttr_ObjNVal);
      cout << std::setw(6) << val;
    }
    cout << endl;
  }

}
catch (GRBException e) {
  cout << "Error code = " << e.getErrorCode() << endl;
  cout << e.getMessage() << endl;
}
catch (...) {
  cout << "Exception during optimization" << endl;
}

// Free environment/vars
```

```
  delete[] Elem;
  delete env;
  return 0;
}
```

params_c++.cpp

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* Use parameters that are associated with a model.

   A MIP is solved for a few seconds with different sets of parameters.
   The one with the smallest MIP gap is selected, and the optimization
   is resumed until the optimal solution is found.
*/

#include "gurobi_c++.h"
using namespace std;

int
main(int argc,
     char *argv[])
{
  if (argc < 2)
  {
    cout << "Usage: params_c++ filename" << endl;
    return 1;
  }

  GRBEnv* env = 0;
  GRBModel *bestModel = 0, *m = 0;
  try
  {
    // Read model and verify that it is a MIP
    env = new GRBEnv();
    m = new GRBModel(*env, argv[1]);
    if (m->get(GRB_IntAttr_IsMIP) == 0)
    {
      cout << "The model is not an integer program" << endl;
      return 1;
    }

    // Set a 2 second time limit
    m->set(GRB_DoubleParam_TimeLimit, 2);

    // Now solve the model with different values of MIPFocus
    bestModel = new GRBModel(*m);
    bestModel->optimize();
    for (int i = 1; i <= 3; ++i)
    {
      m->reset();
```

```
      m->set(GRB_IntParam_MIPFocus, i);
      m->optimize();
      if (bestModel->get(GRB_DoubleAttr_MIPGap) >
                  m->get(GRB_DoubleAttr_MIPGap))
      {
        swap(bestModel, m);
      }
    }

    // Finally, delete the extra model, reset the time limit and
    // continue to solve the best model to optimality
    delete m;
    m = 0;
    bestModel->set(GRB_DoubleParam_TimeLimit, GRB_INFINITY);
    bestModel->optimize();
    cout << "Solved with MIPFocus: " <<
    bestModel->get(GRB_IntParam_MIPFocus) << endl;

  }
  catch (GRBException e)
  {
    cout << "Error code = " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
  }
  catch (...)
  {
    cout << "Error during optimization" << endl;
  }

  delete bestModel;
  delete m;
  delete env;
  return 0;
}
```

piecewise_c++.cpp

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* This example considers the following separable, convex problem:

     minimize    f(x) - y + g(z)
     subject to  x + 2 y + 3 z <= 4
                 x +   y       >= 1
                 x,    y,    z <= 1

   where f(u) = exp(-u) and g(u) = 2 u^2 - 4 u, for all real u. It
   formulates and solves a simpler LP model by approximating f and
   g with piecewise-linear functions. Then it transforms the model
   into a MIP by negating the approximation for f, which corresponds
   to a non-convex piecewise-linear function, and solves it again.
*/

#include "gurobi_c++.h"
#include <cmath>
using namespace std;

double f(double u) { return exp(-u); }
double g(double u) { return 2 * u * u - 4 * u; }

int
main(int   argc,
     char *argv[])
{
  double *ptu = NULL;
  double *ptf = NULL;
  double *ptg = NULL;

  try {

    // Create environment

    GRBEnv env = GRBEnv();

    // Create a new model

    GRBModel model = GRBModel(env);

    // Create variables

    double lb = 0.0, ub = 1.0;
```

```
GRBVar x = model.addVar(lb, ub, 0.0, GRB_CONTINUOUS, "x");
GRBVar y = model.addVar(lb, ub, 0.0, GRB_CONTINUOUS, "y");
GRBVar z = model.addVar(lb, ub, 0.0, GRB_CONTINUOUS, "z");

// Set objective for y

model.setObjective(-y);

// Add piecewise-linear objective functions for x and z

int npts = 101;
ptu = new double[npts];
ptf = new double[npts];
ptg = new double[npts];

for (int i = 0; i < npts; i++) {
  ptu[i] = lb + (ub - lb) * i / (npts - 1);
  ptf[i] = f(ptu[i]);
  ptg[i] = g(ptu[i]);
}

model.setPWLObj(x, npts, ptu, ptf);
model.setPWLObj(z, npts, ptu, ptg);

// Add constraint: x + 2 y + 3 z <= 4

model.addConstr(x + 2 * y + 3 * z <= 4, "c0");

// Add constraint: x + y >= 1

model.addConstr(x + y >= 1, "c1");

// Optimize model as an LP

model.optimize();

cout << "IsMIP: " << model.get(GRB_IntAttr_IsMIP) << endl;

cout << x.get(GRB_StringAttr_VarName) << " "
     << x.get(GRB_DoubleAttr_X) << endl;
cout << y.get(GRB_StringAttr_VarName) << " "
     << y.get(GRB_DoubleAttr_X) << endl;
cout << z.get(GRB_StringAttr_VarName) << " "
     << z.get(GRB_DoubleAttr_X) << endl;
```

```
    cout << "Obj: " << model.get(GRB_DoubleAttr_ObjVal) << endl;

    cout << endl;

    // Negate piecewise-linear objective function for x

    for (int i = 0; i < npts; i++) {
      ptf[i] = -ptf[i];
    }

    model.setPWLObj(x, npts, ptu, ptf);

    // Optimize model as a MIP

    model.optimize();

    cout << "IsMIP: " << model.get(GRB_IntAttr_IsMIP) << endl;

    cout << x.get(GRB_StringAttr_VarName) << " "
         << x.get(GRB_DoubleAttr_X) << endl;
    cout << y.get(GRB_StringAttr_VarName) << " "
         << y.get(GRB_DoubleAttr_X) << endl;
    cout << z.get(GRB_StringAttr_VarName) << " "
         << z.get(GRB_DoubleAttr_X) << endl;

    cout << "Obj: " << model.get(GRB_DoubleAttr_ObjVal) << endl;

  } catch(GRBException e) {
    cout << "Error code = " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
  } catch(...) {
    cout << "Exception during optimization" << endl;
  }

  delete[] ptu;
  delete[] ptf;
  delete[] ptg;

  return 0;
}
```

poolsearch_c++.cpp

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* We find alternative epsilon-optimal solutions to a given knapsack
 * problem by using PoolSearchMode */

#include "gurobi_c++.h"
#include <sstream>
#include <iomanip>
using namespace std;


int main(void)
{
  GRBEnv *env  = 0;
  GRBVar *Elem = 0;
  int e, status, nSolutions;

  try {
    // Sample data
    const int groundSetSize = 10;
    double objCoef[10] =
    {32, 32, 15, 15, 6, 6, 1, 1, 1, 1};
    double knapsackCoef[10] =
    {16, 16,  8,  8, 4, 4, 2, 2, 1, 1};
    double Budget = 33;

    // Create environment
    env = new GRBEnv("poolsearch_c++.log");

    // Create initial model
    GRBModel model = GRBModel(*env);
    model.set(GRB_StringAttr_ModelName, "poolsearch_c++");

    // Initialize decision variables for ground set:
    // x[e] == 1 if element e is chosen
    Elem = model.addVars(groundSetSize, GRB_BINARY);
    model.set(GRB_DoubleAttr_Obj, Elem, objCoef, groundSetSize);

    for (e = 0; e < groundSetSize; e++) {
      ostringstream vname;
      vname << "El" << e;
      Elem[e].set(GRB_StringAttr_VarName, vname.str());
    }
```

```
// Constraint: limit total number of elements to be picked to be at most
// Budget
GRBLinExpr lhs;
lhs = 0;
for (e = 0; e < groundSetSize; e++) {
  lhs += Elem[e] * knapsackCoef[e];
}
model.addConstr(lhs <= Budget, "Budget");

// set global sense for ALL objectives
model.set(GRB_IntAttr_ModelSense, GRB_MAXIMIZE);

// Limit how many solutions to collect
model.set(GRB_IntParam_PoolSolutions, 1024);

// Limit the search space by setting a gap for the worst possible solution that will be acce
model.set(GRB_DoubleParam_PoolGap, 0.10);

// do a systematic search for the k-best solutions
model.set(GRB_IntParam_PoolSearchMode, 2);

// save problem
model.write("poolsearch_c++.lp");

// Optimize
model.optimize();

// Status checking
status = model.get(GRB_IntAttr_Status);

if (status == GRB_INF_OR_UNBD ||
    status == GRB_INFEASIBLE  ||
    status == GRB_UNBOUNDED     ) {
  cout << "The model cannot be solved " <<
         "because it is infeasible or unbounded" << endl;
  return 1;
}
if (status != GRB_OPTIMAL) {
  cout << "Optimization was stopped with status " << status << endl;
  return 1;
}

// Print best selected set
cout << "Selected elements in best solution:" << endl << "\t";
for (e = 0; e < groundSetSize; e++) {
```

```
      if (Elem[e].get(GRB_DoubleAttr_X) < .9) continue;
      cout << " El" << e;
    }
    cout << endl;

    // Print number of solutions stored
    nSolutions = model.get(GRB_IntAttr_SolCount);
    cout << "Number of solutions found: " << nSolutions << endl;

    // Print objective values of solutions
    for (e = 0; e < nSolutions; e++) {
      model.set(GRB_IntParam_SolutionNumber, e);
      cout << model.get(GRB_DoubleAttr_PoolObjVal) << " ";
      if (e%15 == 14) cout << endl;
    }
    cout << endl;

    // print fourth best set if available
    if (nSolutions >= 4) {
      model.set(GRB_IntParam_SolutionNumber, 3);

      cout << "Selected elements in fourth best solution:" << endl << "\t";
      for (e = 0; e < groundSetSize; e++) {
        if (Elem[e].get(GRB_DoubleAttr_Xn) < .9) continue;
        cout << " El" << e;
      }
      cout << endl;
    }
  }
  catch (GRBException e) {
    cout << "Error code = " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
  }
  catch (...) {
    cout << "Exception during optimization" << endl;
  }

  // Free environment/vars
  delete[] Elem;
  delete env;
  return 0;
}
```

sensitivity_c++.cpp

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* A simple sensitivity analysis example which reads a MIP model
   from a file and solves it. Then each binary variable is set
   to 1-X, where X is its value in the optimal solution, and
   the impact on the objective function value is reported.
*/

#include "gurobi_c++.h"
using namespace std;

int
main(int   argc,
     char *argv[])
{
  if (argc < 2) {
    cout << "Usage: sensitivity_c++ filename" << endl;
    return 1;
  }

  GRBVar* vars = NULL;
  double* origX = NULL;

  try {

    // Create environment

    GRBEnv env = GRBEnv();

    // Read and solve model

    GRBModel model = GRBModel(env, argv[1]);

    if (model.get(GRB_IntAttr_IsMIP) == 0) {
      cout << "Model is not a MIP" << endl;
      return 1;
    }

    model.optimize();

    if (model.get(GRB_IntAttr_Status) != GRB_OPTIMAL) {
      cout << "Optimization ended with status "
           << model.get(GRB_IntAttr_Status) << endl;
      return 1;
```

```
}

// Store the optimal solution

double origObjVal = model.get(GRB_DoubleAttr_ObjVal);
vars = model.getVars();
int numVars = model.get(GRB_IntAttr_NumVars);
origX = model.get(GRB_DoubleAttr_X, vars, numVars);

// Disable solver output for subsequent solves

model.set(GRB_IntParam_OutputFlag, 0);

// Iterate through unfixed, binary variables in model

for (int i = 0; i < numVars; i++) {
  GRBVar v = vars[i];
  char vType = v.get(GRB_CharAttr_VType);

  if (v.get(GRB_DoubleAttr_LB) == 0 && v.get(GRB_DoubleAttr_UB) == 1
      && (vType == GRB_BINARY || vType == GRB_INTEGER)) {

    // Set variable to 1-X, where X is its value in optimal solution

    if (origX[i] < 0.5) {
      v.set(GRB_DoubleAttr_LB, 1.0);
      v.set(GRB_DoubleAttr_Start, 1.0);
    } else {
      v.set(GRB_DoubleAttr_UB, 0.0);
      v.set(GRB_DoubleAttr_Start, 0.0);
    }

    // Update MIP start for the other variables

    for (int j = 0; j < numVars; j++) {
      if (j != i) {
        vars[j].set(GRB_DoubleAttr_Start, origX[j]);
      }
    }

    // Solve for new value and capture sensitivity information

    model.optimize();

    if (model.get(GRB_IntAttr_Status) == GRB_OPTIMAL) {
```

```
          cout << "Objective sensitivity for variable "
               << v.get(GRB_StringAttr_VarName) << " is "
               << (model.get(GRB_DoubleAttr_ObjVal) - origObjVal) << endl;
        } else {
          cout << "Objective sensitivity for variable "
               << v.get(GRB_StringAttr_VarName) << " is infinite" << endl;
        }

        // Restore the original variable bounds

        v.set(GRB_DoubleAttr_LB, 0.0);
        v.set(GRB_DoubleAttr_UB, 1.0);
      }
    }

  } catch (GRBException e) {
    cout << "Error code = " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
  } catch (...) {
    cout << "Error during optimization" << endl;
  }

  delete[] vars;
  delete[] origX;

  return 0;
}
```

qcp_c++.cpp

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* This example formulates and solves the following simple QCP model:

     maximize    x
     subject to  x + y + z = 1
                 x^2 + y^2 <= z^2 (second-order cone)
                 x^2 <= yz        (rotated second-order cone)
*/

#include "gurobi_c++.h"
using namespace std;

int
main(int   argc,
     char *argv[])
{
  try {
    GRBEnv env = GRBEnv();

    GRBModel model = GRBModel(env);

    // Create variables

    GRBVar x = model.addVar(0.0, GRB_INFINITY, 0.0, GRB_CONTINUOUS, "x");
    GRBVar y = model.addVar(0.0, GRB_INFINITY, 0.0, GRB_CONTINUOUS, "y");
    GRBVar z = model.addVar(0.0, GRB_INFINITY, 0.0, GRB_CONTINUOUS, "z");

    // Set objective

    GRBLinExpr obj = x;
    model.setObjective(obj, GRB_MAXIMIZE);

    // Add linear constraint: x + y + z = 1

    model.addConstr(x + y + z == 1, "c0");

    // Add second-order cone: x^2 + y^2 <= z^2

    model.addQConstr(x*x + y*y <= z*z, "qc0");

    // Add rotated cone: x^2 <= yz

    model.addQConstr(x*x <= y*z, "qc1");
```

```
    // Optimize model

    model.optimize();

    cout << x.get(GRB_StringAttr_VarName) << " "
         << x.get(GRB_DoubleAttr_X) << endl;
    cout << y.get(GRB_StringAttr_VarName) << " "
         << y.get(GRB_DoubleAttr_X) << endl;
    cout << z.get(GRB_StringAttr_VarName) << " "
         << z.get(GRB_DoubleAttr_X) << endl;

    cout << "Obj: " << model.get(GRB_DoubleAttr_ObjVal) << endl;

  } catch(GRBException e) {
    cout << "Error code = " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
  } catch(...) {
    cout << "Exception during optimization" << endl;
  }

  return 0;
}
```

qp_c++.cpp

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* This example formulates and solves the following simple QP model:

     minimize    x^2 + x*y + y^2 + y*z + z^2 + 2 x
     subject to  x + 2 y + 3 z >= 4
                 x +   y       >= 1

   It solves it once as a continuous model, and once as an integer model.
*/

#include "gurobi_c++.h"
using namespace std;

int
main(int   argc,
     char *argv[])
{
  try {
    GRBEnv env = GRBEnv();

    GRBModel model = GRBModel(env);

    // Create variables

    GRBVar x = model.addVar(0.0, 1.0, 0.0, GRB_CONTINUOUS, "x");
    GRBVar y = model.addVar(0.0, 1.0, 0.0, GRB_CONTINUOUS, "y");
    GRBVar z = model.addVar(0.0, 1.0, 0.0, GRB_CONTINUOUS, "z");

    // Set objective

    GRBQuadExpr obj = x*x + x*y + y*y + y*z + z*z + 2*x;
    model.setObjective(obj);

    // Add constraint: x + 2 y + 3 z >= 4

    model.addConstr(x + 2 * y + 3 * z >= 4, "c0");

    // Add constraint: x + y >= 1

    model.addConstr(x + y >= 1, "c1");

    // Optimize model
```

```
    model.optimize();

    cout << x.get(GRB_StringAttr_VarName) << " "
         << x.get(GRB_DoubleAttr_X) << endl;
    cout << y.get(GRB_StringAttr_VarName) << " "
         << y.get(GRB_DoubleAttr_X) << endl;
    cout << z.get(GRB_StringAttr_VarName) << " "
         << z.get(GRB_DoubleAttr_X) << endl;

    cout << "Obj: " << model.get(GRB_DoubleAttr_ObjVal) << endl;

    // Change variable types to integer

    x.set(GRB_CharAttr_VType, GRB_INTEGER);
    y.set(GRB_CharAttr_VType, GRB_INTEGER);
    z.set(GRB_CharAttr_VType, GRB_INTEGER);

    // Optimize model

    model.optimize();

    cout << x.get(GRB_StringAttr_VarName) << " "
         << x.get(GRB_DoubleAttr_X) << endl;
    cout << y.get(GRB_StringAttr_VarName) << " "
         << y.get(GRB_DoubleAttr_X) << endl;
    cout << z.get(GRB_StringAttr_VarName) << " "
         << z.get(GRB_DoubleAttr_X) << endl;

    cout << "Obj: " << model.get(GRB_DoubleAttr_ObjVal) << endl;

  } catch(GRBException e) {
    cout << "Error code = " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
  } catch(...) {
    cout << "Exception during optimization" << endl;
  }

  return 0;
}
```

sos_c++.cpp

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* This example creates a very simple Special Ordered Set (SOS) model.
   The model consists of 3 continuous variables, no linear constraints,
   and a pair of SOS constraints of type 1. */

#include "gurobi_c++.h"
using namespace std;

int
main(int   argc,
     char *argv[])
{
  GRBEnv *env = 0;
  GRBVar *x = 0;
  try {
    env = new GRBEnv();

    GRBModel model = GRBModel(*env);

    // Create variables

    double ub[]    = {1, 1, 2};
    double obj[]   = {-2, -1, -1};
    string names[] = {"x0", "x1", "x2"};

    x = model.addVars(NULL, ub, obj, NULL, names, 3);

    // Add first SOS1: x0=0 or x1=0

    GRBVar sosv1[]  = {x[0], x[1]};
    double soswt1[] = {1, 2};

    model.addSOS(sosv1, soswt1, 2, GRB_SOS_TYPE1);

    // Add second SOS1: x0=0 or x2=0 */

    GRBVar sosv2[]  = {x[0], x[2]};
    double soswt2[] = {1, 2};

    model.addSOS(sosv2, soswt2, 2, GRB_SOS_TYPE1);

    // Optimize model
```

```
    model.optimize();

    for (int i = 0; i < 3; i++)
      cout << x[i].get(GRB_StringAttr_VarName) << " "
           << x[i].get(GRB_DoubleAttr_X) << endl;

    cout << "Obj: " << model.get(GRB_DoubleAttr_ObjVal) << endl;

  } catch(GRBException e) {
    cout << "Error code = " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
  } catch(...) {
    cout << "Exception during optimization" << endl;
  }

  delete[] x;
  delete env;
  return 0;
}
```

sudoku_c++.cpp

```
/* Copyright 2017, Gurobi Optimization, Inc. */
/*
  Sudoku example.

  The Sudoku board is a 9x9 grid, which is further divided into a 3x3 grid
  of 3x3 grids.  Each cell in the grid must take a value from 0 to 9.
  No two grid cells in the same row, column, or 3x3 subgrid may take the
  same value.

  In the MIP formulation, binary variables x[i,j,v] indicate whether
  cell <i,j> takes value 'v'.  The constraints are as follows:
    1. Each cell must take exactly one value (sum_v x[i,j,v] = 1)
    2. Each value is used exactly once per row (sum_i x[i,j,v] = 1)
    3. Each value is used exactly once per column (sum_j x[i,j,v] = 1)
    4. Each value is used exactly once per 3x3 subgrid (sum_grid x[i,j,v] = 1)

  Input datasets for this example can be found in examples/data/sudoku*.
*/

#include "gurobi_c++.h"
#include <sstream>
using namespace std;

#define sd 3
#define n (sd*sd)

string itos(int i) {stringstream s; s << i; return s.str(); }

int
main(int   argc,
     char *argv[])
{
  try {
    GRBEnv env = GRBEnv();
    GRBModel model = GRBModel(env);

    GRBVar vars[n][n][n];
    int i, j, v;

    // Create 3-D array of model variables

    for (i = 0; i < n; i++) {
      for (j = 0; j < n; j++) {
        for (v = 0; v < n; v++) {
```

```
      string s = "G_" + itos(i) + "_" + itos(j) + "_" + itos(v);
      vars[i][j][v] = model.addVar(0.0, 1.0, 0.0, GRB_BINARY, s);
    }
  }
}

// Add constraints

// Each cell must take one value

for (i = 0; i < n; i++) {
  for (j = 0; j < n; j++) {
    GRBLinExpr expr = 0;
    for (v = 0; v < n; v++)
      expr += vars[i][j][v];
    string s = "V_" + itos(i) + "_" + itos(j);
    model.addConstr(expr, GRB_EQUAL, 1.0, s);
  }
}

// Each value appears once per row

for (i = 0; i < n; i++) {
  for (v = 0; v < n; v++) {
    GRBLinExpr expr = 0;
    for (j = 0; j < n; j++)
      expr += vars[i][j][v];
    string s = "R_" + itos(i) + "_" + itos(v);
    model.addConstr(expr == 1.0, s);
  }
}

// Each value appears once per column

for (j = 0; j < n; j++) {
  for (v = 0; v < n; v++) {
    GRBLinExpr expr = 0;
    for (i = 0; i < n; i++)
      expr += vars[i][j][v];
    string s = "C_" + itos(j) + "_" + itos(v);
    model.addConstr(expr == 1.0, s);
  }
}

// Each value appears once per sub-grid
```

```
for (v = 0; v < n; v++) {
  for (int i0 = 0; i0 < sd; i0++) {
    for (int j0 = 0; j0 < sd; j0++) {
      GRBLinExpr expr = 0;
      for (int i1 = 0; i1 < sd; i1++) {
        for (int j1 = 0; j1 < sd; j1++) {
          expr += vars[i0*sd+i1][j0*sd+j1][v];
        }
      }

      string s = "Sub_" + itos(v) + "_" + itos(i0) + "_" + itos(j0);
      model.addConstr(expr == 1.0, s);
    }
  }
}

// Fix variables associated with pre-specified cells

char input[10];
for (i = 0; i < n; i++) {
  cin >> input;
  for (j = 0; j < n; j++) {
    int val = (int) input[j] - 48 - 1; // 0-based

    if (val >= 0)
      vars[i][j][val].set(GRB_DoubleAttr_LB, 1.0);
  }
}

// Optimize model

model.optimize();

// Write model to file

model.write("sudoku.lp");

cout << endl;
for (i = 0; i < n; i++) {
  for (j = 0; j < n; j++) {
    for (v = 0; v < n; v++) {
      if (vars[i][j][v].get(GRB_DoubleAttr_X) > 0.5)
        cout << v+1;
    }
```

```
      }
      cout << endl;
    }
    cout << endl;
  } catch(GRBException e) {
    cout << "Error code = " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
  } catch (...) {
    cout << "Error during optimization" << endl;
  }

  return 0;
}
```

tsp_c++.cpp

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* Solve a traveling salesman problem on a randomly generated set of
   points using lazy constraints.   The base MIP model only includes
   'degree-2' constraints, requiring each node to have exactly
   two incident edges.  Solutions to this model may contain subtours -
   tours that don't visit every node.  The lazy constraint callback
   adds new constraints to cut them off. */

#include "gurobi_c++.h"
#include <cassert>
#include <cstdlib>
#include <cmath>
#include <sstream>
using namespace std;

string itos(int i) {stringstream s; s << i; return s.str(); }
double distance(double* x, double* y, int i, int j);
void findsubtour(int n, double** sol, int* tourlenP, int* tour);

// Subtour elimination callback.  Whenever a feasible solution is found,
// find the smallest subtour, and add a subtour elimination constraint
// if the tour doesn't visit every node.

class subtourelim: public GRBCallback
{
  public:
    GRBVar** vars;
    int n;
    subtourelim(GRBVar** xvars, int xn) {
      vars = xvars;
      n    = xn;
    }
  protected:
    void callback() {
      try {
        if (where == GRB_CB_MIPSOL) {
          // Found an integer feasible solution - does it visit every node?
          double **x = new double*[n];
          int *tour = new int[n];
          int i, j, len;
          for (i = 0; i < n; i++)
            x[i] = getSolution(vars[i], n);
```

```
          findsubtour(n, x, &len, tour);

          if (len < n) {
            // Add subtour elimination constraint
            GRBLinExpr expr = 0;
            for (i = 0; i < len; i++)
              for (j = i+1; j < len; j++)
                expr += vars[tour[i]][tour[j]];
            addLazy(expr <= len-1);
          }

          for (i = 0; i < n; i++)
            delete[] x[i];
          delete[] x;
          delete[] tour;
        }
      } catch (GRBException e) {
        cout << "Error number: " << e.getErrorCode() << endl;
        cout << e.getMessage() << endl;
      } catch (...) {
        cout << "Error during callback" << endl;
      }
    }
};

// Given an integer-feasible solution 'sol', find the smallest
// sub-tour.  Result is returned in 'tour', and length is
// returned in 'tourlenP'.

void
findsubtour(int      n,
            double** sol,
            int*     tourlenP,
            int*     tour)
{
  bool* seen = new bool[n];
  int bestind, bestlen;
  int i, node, len, start;

  for (i = 0; i < n; i++)
    seen[i] = false;

  start = 0;
  bestlen = n+1;
  bestind = -1;
```

```
  node = 0;
  while (start < n) {
    for (node = 0; node < n; node++)
      if (!seen[node])
        break;
    if (node == n)
      break;
    for (len = 0; len < n; len++) {
      tour[start+len] = node;
      seen[node] = true;
      for (i = 0; i < n; i++) {
        if (sol[node][i] > 0.5 && !seen[i]) {
          node = i;
          break;
        }
      }
      if (i == n) {
        len++;
        if (len < bestlen) {
          bestlen = len;
          bestind = start;
        }
        start += len;
        break;
      }
    }
  }

  for (i = 0; i < bestlen; i++)
    tour[i] = tour[bestind+i];
  *tourlenP = bestlen;

  delete[] seen;
}

// Euclidean distance between points 'i' and 'j'.

double
distance(double* x,
         double* y,
         int     i,
         int     j)
{
  double dx = x[i]-x[j];
  double dy = y[i]-y[j];
```

```
  return sqrt(dx*dx+dy*dy);
}

int
main(int   argc,
     char *argv[])
{
  if (argc < 2) {
    cout << "Usage: tsp_c++ size" << endl;
    return 1;
  }

  int n = atoi(argv[1]);
  double* x = new double[n];
  double* y = new double[n];

  int i;
  for (i = 0; i < n; i++) {
    x[i] = ((double) rand())/RAND_MAX;
    y[i] = ((double) rand())/RAND_MAX;
  }

  GRBEnv *env = NULL;
  GRBVar **vars = NULL;

  vars = new GRBVar*[n];
  for (i = 0; i < n; i++)
    vars[i] = new GRBVar[n];

  try {
    int j;

    env = new GRBEnv();
    GRBModel model = GRBModel(*env);

    // Must set LazyConstraints parameter when using lazy constraints

    model.set(GRB_IntParam_LazyConstraints, 1);

    // Create binary decision variables

    for (i = 0; i < n; i++) {
      for (j = 0; j <= i; j++) {
        vars[i][j] = model.addVar(0.0, 1.0, distance(x, y, i, j),
```

```
                                GRB_BINARY, "x_"+itos(i)+"_"+itos(j));
    vars[j][i] = vars[i][j];
  }
}

// Degree-2 constraints

for (i = 0; i < n; i++) {
  GRBLinExpr expr = 0;
  for (j = 0; j < n; j++)
    expr += vars[i][j];
  model.addConstr(expr == 2, "deg2_"+itos(i));
}

// Forbid edge from node back to itself

for (i = 0; i < n; i++)
  vars[i][i].set(GRB_DoubleAttr_UB, 0);

// Set callback function

subtourelim cb = subtourelim(vars, n);
model.setCallback(&cb);

// Optimize model

model.optimize();

// Extract solution

if (model.get(GRB_IntAttr_SolCount) > 0) {
  double **sol = new double*[n];
  for (i = 0; i < n; i++)
    sol[i] = model.get(GRB_DoubleAttr_X, vars[i], n);

  int* tour = new int[n];
  int len;

  findsubtour(n, sol, &len, tour);
  assert(len == n);

  cout << "Tour: ";
  for (i = 0; i < len; i++)
    cout << tour[i] << " ";
  cout << endl;
```

```
      for (i = 0; i < n; i++)
        delete[] sol[i];
      delete[] sol;
      delete[] tour;
    }

  } catch (GRBException e) {
    cout << "Error number: " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
  } catch (...) {
    cout << "Error during optimization" << endl;
  }

  for (i = 0; i < n; i++)
    delete[] vars[i];
  delete[] vars;
  delete[] x;
  delete[] y;
  delete env;
  return 0;
}
```

tune_c++.cpp

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* This example reads a model from a file and tunes it.
   It then writes the best parameter settings to a file
   and solves the model using these parameters. */

#include "gurobi_c++.h"
#include <cmath>
using namespace std;

int
main(int   argc,
     char *argv[])
{
  if (argc < 2) {
    cout << "Usage: tune_c++ filename" << endl;
    return 1;
  }

  GRBEnv *env = 0;
  try {
    env = new GRBEnv();

    // Read model from file

    GRBModel model = GRBModel(*env, argv[1]);

    // Set the TuneResults parameter to 1

    model.set(GRB_IntParam_TuneResults, 1);

    // Tune the model

    model.tune();

    // Get the number of tuning results

    int resultcount = model.get(GRB_IntAttr_TuneResultCount);

    if (resultcount > 0) {

      // Load the tuned parameters into the model's environment

      model.getTuneResult(0);
```

```
      // Write tuned parameters to a file

      model.write("tune.prm");

      // Solve the model using the tuned parameters

      model.optimize();
    }
  } catch(GRBException e) {
    cout << "Error code = " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
  } catch (...) {
    cout << "Error during tuning" << endl;
  }

  delete env;
  return 0;
}
```

workforce1_c++.cpp

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* Assign workers to shifts; each worker may or may not be available on a
   particular day. If the problem cannot be solved, use IIS to find a set of
   conflicting constraints. Note that there may be additional conflicts
   besides what is reported via IIS. */

#include "gurobi_c++.h"
#include <sstream>
using namespace std;

int
main(int argc,
     char *argv[])
{
  GRBEnv* env = 0;
  GRBConstr* c = 0;
  GRBVar** x = 0;
  int xCt = 0;
  try
  {

    // Sample data
    const int nShifts = 14;
    const int nWorkers = 7;

    // Sets of days and workers
    string Shifts[] =
      { "Mon1", "Tue2", "Wed3", "Thu4", "Fri5", "Sat6",
        "Sun7", "Mon8", "Tue9", "Wed10", "Thu11", "Fri12", "Sat13",
        "Sun14" };
    string Workers[] =
      { "Amy", "Bob", "Cathy", "Dan", "Ed", "Fred", "Gu" };

    // Number of workers required for each shift
    double shiftRequirements[] =
      { 3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5 };

    // Amount each worker is paid to work one shift
    double pay[] = { 10, 12, 10, 8, 8, 9, 11 };

    // Worker availability: 0 if the worker is unavailable for a shift
    double availability[][nShifts] =
      { { 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1 },
```

```
    { 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0 },
    { 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
    { 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1 },
    { 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1 },
    { 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1 },
    { 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 } };

// Model
env = new GRBEnv();
GRBModel model = GRBModel(*env);
model.set(GRB_StringAttr_ModelName, "assignment");

// Assignment variables: x[w][s] == 1 if worker w is assigned
// to shift s. Since an assignment model always produces integer
// solutions, we use continuous variables and solve as an LP.
x = new GRBVar*[nWorkers];
for (int w = 0; w < nWorkers; ++w)
{
  x[w] = model.addVars(nShifts);
  xCt++;
  for (int s = 0; s < nShifts; ++s)
  {
    ostringstream vname;
    vname << Workers[w] << "." << Shifts[s];
    x[w][s].set(GRB_DoubleAttr_UB, availability[w][s]);
    x[w][s].set(GRB_DoubleAttr_Obj, pay[w]);
    x[w][s].set(GRB_StringAttr_VarName, vname.str());
  }
}

// The objective is to minimize the total pay costs
model.set(GRB_IntAttr_ModelSense, GRB_MINIMIZE);

// Constraint: assign exactly shiftRequirements[s] workers
// to each shift s
for (int s = 0; s < nShifts; ++s)
{
  GRBLinExpr lhs = 0;
  for (int w = 0; w < nWorkers; ++w)
  {
    lhs += x[w][s];
  }
  model.addConstr(lhs == shiftRequirements[s], Shifts[s]);
}
```

```
  // Optimize
  model.optimize();
  int status = model.get(GRB_IntAttr_Status);
  if (status == GRB_UNBOUNDED)
  {
    cout << "The model cannot be solved "
    << "because it is unbounded" << endl;
    return 1;
  }
  if (status == GRB_OPTIMAL)
  {
    cout << "The optimal objective is " <<
    model.get(GRB_DoubleAttr_ObjVal) << endl;
    return 0;
  }
  if ((status != GRB_INF_OR_UNBD) && (status != GRB_INFEASIBLE))
  {
    cout << "Optimization was stopped with status " << status << endl;
    return 1;
  }

  // do IIS
  cout << "The model is infeasible; computing IIS" << endl;
  model.computeIIS();
  cout << "\nThe following constraint(s) "
  << "cannot be satisfied:" << endl;
  c = model.getConstrs();
  for (int i = 0; i < model.get(GRB_IntAttr_NumConstrs); ++i)
  {
    if (c[i].get(GRB_IntAttr_IISConstr) == 1)
    {
      cout << c[i].get(GRB_StringAttr_ConstrName) << endl;
    }
  }

}
catch (GRBException e)
{
  cout << "Error code = " << e.getErrorCode() << endl;
  cout << e.getMessage() << endl;
}
catch (...)
{
  cout << "Exception during optimization" << endl;
}
```

```
  delete[] c;
  for (int i = 0; i < xCt; ++i) {
    delete[] x[i];
  }
  delete[] x;
  delete env;
  return 0;
}
```

workforce2_c++.cpp

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* Assign workers to shifts; each worker may or may not be available on a
   particular day. If the problem cannot be solved, use IIS iteratively to
   find all conflicting constraints. */

#include "gurobi_c++.h"
#include <sstream>
#include <deque>
using namespace std;

int
main(int argc,
     char *argv[])
{
  GRBEnv* env = 0;
  GRBConstr* c = 0;
  GRBVar** x = 0;
  int xCt = 0;
  try
  {

    // Sample data
    const int nShifts = 14;
    const int nWorkers = 7;

    // Sets of days and workers
    string Shifts[] =
      { "Mon1", "Tue2", "Wed3", "Thu4", "Fri5", "Sat6",
        "Sun7", "Mon8", "Tue9", "Wed10", "Thu11", "Fri12", "Sat13",
        "Sun14" };
    string Workers[] =
      { "Amy", "Bob", "Cathy", "Dan", "Ed", "Fred", "Gu" };

    // Number of workers required for each shift
    double shiftRequirements[] =
      { 3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5 };

    // Amount each worker is paid to work one shift
    double pay[] = { 10, 12, 10, 8, 8, 9, 11 };

    // Worker availability: 0 if the worker is unavailable for a shift
    double availability[][nShifts] =
      { { 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1 },
```

```
    { 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0 },
    { 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
    { 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1 },
    { 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1 },
    { 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1 },
    { 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 } };

// Model
env = new GRBEnv();
GRBModel model = GRBModel(*env);
model.set(GRB_StringAttr_ModelName, "assignment");

// Assignment variables: x[w][s] == 1 if worker w is assigned
// to shift s. Since an assignment model always produces integer
// solutions, we use continuous variables and solve as an LP.
x = new GRBVar*[nWorkers];
for (int w = 0; w < nWorkers; ++w)
{
  x[w] = model.addVars(nShifts);
  xCt++;
  for (int s = 0; s < nShifts; ++s)
  {
    ostringstream vname;
    vname << Workers[w] << "." << Shifts[s];
    x[w][s].set(GRB_DoubleAttr_UB, availability[w][s]);
    x[w][s].set(GRB_DoubleAttr_Obj, pay[w]);
    x[w][s].set(GRB_StringAttr_VarName, vname.str());
  }
}

// The objective is to minimize the total pay costs
model.set(GRB_IntAttr_ModelSense, GRB_MINIMIZE);

// Constraint: assign exactly shiftRequirements[s] workers
// to each shift s
for (int s = 0; s < nShifts; ++s)
{
  GRBLinExpr lhs = 0;
  for (int w = 0; w < nWorkers; ++w)
  {
    lhs += x[w][s];
  }
  model.addConstr(lhs == shiftRequirements[s], Shifts[s]);
}
```

```
// Optimize
model.optimize();
int status = model.get(GRB_IntAttr_Status);
if (status == GRB_UNBOUNDED)
{
  cout << "The model cannot be solved "
  << "because it is unbounded" << endl;
  return 1;
}
if (status == GRB_OPTIMAL)
{
  cout << "The optimal objective is " <<
  model.get(GRB_DoubleAttr_ObjVal) << endl;
  return 0;
}
if ((status != GRB_INF_OR_UNBD) && (status != GRB_INFEASIBLE))
{
  cout << "Optimization was stopped with status " << status << endl;
  return 1;
}

// do IIS
cout << "The model is infeasible; computing IIS" << endl;
deque<string> removed;

// Loop until we reduce to a model that can be solved
while (1)
{
  model.computeIIS();
  cout << "\nThe following constraint cannot be satisfied:" << endl;
  c = model.getConstrs();
  for (int i = 0; i < model.get(GRB_IntAttr_NumConstrs); ++i)
  {
    if (c[i].get(GRB_IntAttr_IISConstr) == 1)
    {
      cout << c[i].get(GRB_StringAttr_ConstrName) << endl;
      // Remove a single constraint from the model
      removed.push_back(c[i].get(GRB_StringAttr_ConstrName));
      model.remove(c[i]);
      break;
    }
  }
  delete[] c;
  c = 0;
```

```
      cout << endl;
      model.optimize();
      status = model.get(GRB_IntAttr_Status);

      if (status == GRB_UNBOUNDED)
      {
        cout << "The model cannot be solved because it is unbounded" << endl;
        return 0;
      }
      if (status == GRB_OPTIMAL)
      {
        break;
      }
      if ((status != GRB_INF_OR_UNBD) && (status != GRB_INFEASIBLE))
      {
        cout << "Optimization was stopped with status " << status << endl;
        return 1;
      }
    }
    cout << "\nThe following constraints were removed "
    << "to get a feasible LP:" << endl;

    for (deque<string>::iterator r = removed.begin();
         r != removed.end();
         ++r)
    {
      cout << *r << " ";
    }
    cout << endl;

  }
  catch (GRBException e)
  {
    cout << "Error code = " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
  }
  catch (...)
  {
    cout << "Exception during optimization" << endl;
  }

  delete[] c;
  for (int i = 0; i < xCt; ++i) {
    delete[] x[i];
  }
```

```
  delete[] x;
  delete env;
  return 0;
}
```

workforce3_c++.cpp

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* Assign workers to shifts; each worker may or may not be available on a
   particular day. If the problem cannot be solved, relax the model
   to determine which constraints cannot be satisfied, and how much
   they need to be relaxed. */

#include "gurobi_c++.h"
#include <sstream>
using namespace std;

int
main(int argc,
     char *argv[])
{
  GRBEnv* env = 0;
  GRBConstr* c = 0;
  GRBVar** x = 0;
  GRBVar* vars = 0;
  int xCt = 0;
  try
  {

    // Sample data
    const int nShifts = 14;
    const int nWorkers = 7;

    // Sets of days and workers
    string Shifts[] =
      { "Mon1", "Tue2", "Wed3", "Thu4", "Fri5", "Sat6",
        "Sun7", "Mon8", "Tue9", "Wed10", "Thu11", "Fri12", "Sat13",
        "Sun14" };
    string Workers[] =
      { "Amy", "Bob", "Cathy", "Dan", "Ed", "Fred", "Gu" };

    // Number of workers required for each shift
    double shiftRequirements[] =
      { 3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5 };

    // Amount each worker is paid to work one shift
    double pay[] = { 10, 12, 10, 8, 8, 9, 11 };

    // Worker availability: 0 if the worker is unavailable for a shift
    double availability[][nShifts] =
```

```
  { { 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1 },
    { 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0 },
    { 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
    { 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1 },
    { 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1 },
    { 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1 },
    { 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 } };

// Model
env = new GRBEnv();
GRBModel model = GRBModel(*env);
model.set(GRB_StringAttr_ModelName, "assignment");

// Assignment variables: x[w][s] == 1 if worker w is assigned
// to shift s. Since an assignment model always produces integer
// solutions, we use continuous variables and solve as an LP.
x = new GRBVar*[nWorkers];
for (int w = 0; w < nWorkers; ++w)
{
  x[w] = model.addVars(nShifts);
  xCt++;
  for (int s = 0; s < nShifts; ++s)
  {
    ostringstream vname;
    vname << Workers[w] << "." << Shifts[s];
    x[w][s].set(GRB_DoubleAttr_UB, availability[w][s]);
    x[w][s].set(GRB_DoubleAttr_Obj, pay[w]);
    x[w][s].set(GRB_StringAttr_VarName, vname.str());
  }
}

// The objective is to minimize the total pay costs
model.set(GRB_IntAttr_ModelSense, GRB_MINIMIZE);

// Constraint: assign exactly shiftRequirements[s] workers
// to each shift s
for (int s = 0; s < nShifts; ++s)
{
  GRBLinExpr lhs = 0;
  for (int w = 0; w < nWorkers; ++w)
  {
    lhs += x[w][s];
  }
  model.addConstr(lhs == shiftRequirements[s], Shifts[s]);
}
```

```
// Optimize
model.optimize();
int status = model.get(GRB_IntAttr_Status);
if (status == GRB_UNBOUNDED)
{
  cout << "The model cannot be solved "
  << "because it is unbounded" << endl;
  return 1;
}
if (status == GRB_OPTIMAL)
{
  cout << "The optimal objective is " <<
  model.get(GRB_DoubleAttr_ObjVal) << endl;
  return 0;
}
if ((status != GRB_INF_OR_UNBD) && (status != GRB_INFEASIBLE))
{
  cout << "Optimization was stopped with status " << status << endl;
  return 1;
}

// Relax the constraints to make the model feasible
cout << "The model is infeasible; relaxing the constraints" << endl;
int orignumvars = model.get(GRB_IntAttr_NumVars);
model.feasRelax(0, false, false, true);
model.optimize();
status = model.get(GRB_IntAttr_Status);
if ((status == GRB_INF_OR_UNBD) || (status == GRB_INFEASIBLE) ||
    (status == GRB_UNBOUNDED))
{
  cout << "The relaxed model cannot be solved " <<
  "because it is infeasible or unbounded" << endl;
  return 1;
}
if (status != GRB_OPTIMAL)
{
  cout << "Optimization was stopped with status " << status << endl;
  return 1;
}

cout << "\nSlack values:" << endl;
vars = model.getVars();
for (int i = orignumvars; i < model.get(GRB_IntAttr_NumVars); ++i)
{
```

```
      GRBVar sv = vars[i];
      if (sv.get(GRB_DoubleAttr_X) > 1e-6)
      {
        cout << sv.get(GRB_StringAttr_VarName) << " = " <<
        sv.get(GRB_DoubleAttr_X) << endl;
      }
    }

  }
  catch (GRBException e)
  {
    cout << "Error code = " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
  }
  catch (...)
  {
    cout << "Exception during optimization" << endl;
  }

  delete[] c;
  for (int i = 0; i < xCt; ++i) {
    delete[] x[i];
  }
  delete[] x;
  delete[] vars;
  delete env;
  return 0;
}
```

workforce4_c++.cpp

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* Assign workers to shifts; each worker may or may not be available on a
 * particular day. We use Pareto optimization to solve the model:
 * first, we minimize the linear sum of the slacks. Then, we constrain
 * the sum of the slacks, and we minimize a quadratic objective that
 * tries to balance the workload among the workers. */

#include "gurobi_c++.h"
#include <sstream>
using namespace std;

int solveAndPrint(GRBModel& model, GRBVar& totSlack,
                  int nWorkers, string* Workers,
                  GRBVar* totShifts) throw(GRBException);

int
main(int   argc,
     char *argv[])
{
  GRBEnv* env = 0;
  GRBVar** x = 0;
  GRBVar* slacks = 0;
  GRBVar* totShifts = 0;
  GRBVar* diffShifts = 0;
  int xCt = 0;

  try
  {
    // Sample data
    const int nShifts = 14;
    const int nWorkers = 7;

    // Sets of days and workers
    string Shifts[] =
      { "Mon1", "Tue2", "Wed3", "Thu4", "Fri5", "Sat6",
        "Sun7", "Mon8", "Tue9", "Wed10", "Thu11", "Fri12", "Sat13",
        "Sun14" };
    string Workers[] =
      { "Amy", "Bob", "Cathy", "Dan", "Ed", "Fred", "Gu" };

    // Number of workers required for each shift
    double shiftRequirements[] =
      { 3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5 };
```

```
// Worker availability: 0 if the worker is unavailable for a shift
double availability[][nShifts] =
  { { 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1 },
    { 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0 },
    { 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
    { 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1 },
    { 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1 },
    { 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1 },
    { 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 } };

// Model
env = new GRBEnv();
GRBModel model = GRBModel(*env);
model.set(GRB_StringAttr_ModelName, "assignment");

// Assignment variables: x[w][s] == 1 if worker w is assigned
// to shift s. This is no longer a pure assignment model, so we must
// use binary variables.
x = new GRBVar*[nWorkers];
int s, w;

for (w = 0; w < nWorkers; ++w) {
  x[w] = model.addVars(nShifts);
  xCt++;

  for (s = 0; s < nShifts; ++s) {
    ostringstream vname;

    vname << Workers[w] << "." << Shifts[s];
    x[w][s].set(GRB_DoubleAttr_UB, availability[w][s]);
    x[w][s].set(GRB_CharAttr_VType, GRB_BINARY);
    x[w][s].set(GRB_StringAttr_VarName, vname.str());
  }
}

// Slack variables for each shift constraint so that the shifts can
// be satisfied
slacks = model.addVars(nShifts);
for (s = 0; s < nShifts; ++s) {
  ostringstream vname;

  vname << Shifts[s] << "Slack";
  slacks[s].set(GRB_StringAttr_VarName, vname.str());
}
```

```
// Variable to represent the total slack
GRBVar totSlack = model.addVar(0, GRB_INFINITY, 0, GRB_CONTINUOUS,
                               "totSlack");

// Variables to count the total shifts worked by each worker
totShifts = model.addVars(nWorkers);
for (w = 0; w < nWorkers; ++w) {
  ostringstream vname;

  vname << Workers[w] << "TotShifts";
  totShifts[w].set(GRB_StringAttr_VarName, vname.str());
}

GRBLinExpr lhs;

// Constraint: assign exactly shiftRequirements[s] workers
// to each shift s
for (s = 0; s < nShifts; ++s) {
  lhs = 0;
  lhs += slacks[s];

  for (w = 0; w < nWorkers; ++w) {
    lhs += x[w][s];
  }

  model.addConstr(lhs == shiftRequirements[s], Shifts[s]);
}

// Constraint: set totSlack equal to the total slack
lhs = 0;
for (s = 0; s < nShifts; ++s)
{
  lhs += slacks[s];
}
model.addConstr(lhs == totSlack, "totSlack");

// Constraint: compute the total number of shifts for each worker
for (w = 0; w < nWorkers; ++w) {
  lhs = 0;
  for (s = 0; s < nShifts; ++s) {
    lhs += x[w][s];
  }
  ostringstream vname;
  vname << "totShifts" << Workers[w];
```

```
  model.addConstr(lhs == totShifts[w], vname.str());
}

// Objective: minimize the total slack
GRBLinExpr obj = 0;
obj += totSlack;
model.setObjective(obj);

// Optimize
int status = solveAndPrint(model, totSlack, nWorkers, Workers, totShifts);
if (status != GRB_OPTIMAL)
  return 1;

// Constrain the slack by setting its upper and lower bounds
totSlack.set(GRB_DoubleAttr_UB, totSlack.get(GRB_DoubleAttr_X));
totSlack.set(GRB_DoubleAttr_LB, totSlack.get(GRB_DoubleAttr_X));

// Variable to count the average number of shifts worked
GRBVar avgShifts =
  model.addVar(0, GRB_INFINITY, 0, GRB_CONTINUOUS, "avgShifts");

// Variables to count the difference from average for each worker;
// note that these variables can take negative values.
diffShifts = model.addVars(nWorkers);
for (w = 0; w < nWorkers; ++w) {
  ostringstream vname;
  vname << Workers[w] << "Diff";
  diffShifts[w].set(GRB_StringAttr_VarName, vname.str());
  diffShifts[w].set(GRB_DoubleAttr_LB, -GRB_INFINITY);
}

// Constraint: compute the average number of shifts worked
lhs = 0;
for (w = 0; w < nWorkers; ++w) {
  lhs += totShifts[w];
}
model.addConstr(lhs == nWorkers * avgShifts, "avgShifts");

// Constraint: compute the difference from the average number of shifts
for (w = 0; w < nWorkers; ++w) {
  lhs = 0;
  lhs += totShifts[w];
  lhs -= avgShifts;
  ostringstream vname;
  vname << Workers[w] << "Diff";
```

```
      model.addConstr(lhs == diffShifts[w], vname.str());
    }

    // Objective: minimize the sum of the square of the difference from the
    // average number of shifts worked
    GRBQuadExpr qobj;
    for (w = 0; w < nWorkers; ++w) {
      qobj += diffShifts[w] * diffShifts[w];
    }
    model.setObjective(qobj);

    // Optimize
    status = solveAndPrint(model, totSlack, nWorkers, Workers, totShifts);
    if (status != GRB_OPTIMAL)
      return 1;
  }
  catch (GRBException e) {
    cout << "Error code = " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
  }
  catch (...) {
    cout << "Exception during optimization" << endl;
  }

  for (int i = 0; i < xCt; ++i) {
    delete[] x[i];
  }
  delete[] x;
  delete[] slacks;
  delete[] totShifts;
  delete[] diffShifts;
  delete env;

  return 0;
}

int solveAndPrint(GRBModel& model,
                  GRBVar&   totSlack,
                  int       nWorkers,
                  string*   Workers,
                  GRBVar*   totShifts) throw(GRBException)
{
  model.optimize();
  int status = model.get(GRB_IntAttr_Status);
```

```
  if ((status == GRB_INF_OR_UNBD) ||
      (status == GRB_INFEASIBLE)  ||
      (status == GRB_UNBOUNDED)     ) {
    cout << "The model cannot be solved " <<
    "because it is infeasible or unbounded" << endl;
    return status;
  }
  if (status != GRB_OPTIMAL) {
    cout << "Optimization was stopped with status " << status << endl;
    return status;
  }

  // Print total slack and the number of shifts worked for each worker
  cout << endl << "Total slack required: " <<
    totSlack.get(GRB_DoubleAttr_X) << endl;
  for (int w = 0; w < nWorkers; ++w) {
    cout << Workers[w] << " worked " <<
    totShifts[w].get(GRB_DoubleAttr_X) << " shifts" << endl;
  }
  cout << endl;

  return status;
}
```

workforce5_c++.cpp

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* Assign workers to shifts; each worker may or may not be available on a
   particular day. We use multi-objective optimization to solve the model.
   The highest-priority objective minimizes the sum of the slacks
   (i.e., the total number of uncovered shifts). The secondary objective
   minimizes the difference between the maximum and minimum number of
   shifts worked among all workers.  The second optimization is allowed
   to degrade the first objective by up to the smaller value of 10% and 2 */

#include "gurobi_c++.h"
#include <sstream>
using namespace std;

int solveAndPrint(GRBModel& model, GRBVar& totSlack,
                  int nWorkers, string* Workers,
                  GRBVar* totShifts) throw(GRBException);

int
main(int   argc,
     char *argv[])
{
  GRBEnv  *env       = 0;
  GRBVar **x         = 0;
  GRBVar  *slacks    = 0;
  GRBVar  *totShifts = 0;
  int      xCt       = 0;
  int      s, w;

  try {
    // Sample data
    const int nShifts = 14;
    const int nWorkers = 8;

    // Sets of days and workers
    string Shifts[] =
    { "Mon1", "Tue2", "Wed3", "Thu4", "Fri5", "Sat6",
      "Sun7", "Mon8", "Tue9", "Wed10", "Thu11", "Fri12", "Sat13",
      "Sun14" };
    string Workers[] =
    { "Amy", "Bob", "Cathy", "Dan", "Ed", "Fred", "Gu", "Tobi" };

    // Number of workers required for each shift
    double shiftRequirements[] =
```

```
{ 3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5 };

// Worker availability: 0 if the worker is unavailable for a shift
double availability[][14] =
{ { 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1 },
  { 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0 },
  { 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
  { 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1 },
  { 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1 },
  { 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1 },
  { 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1 },
  { 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 } };

// Create environment
env = new GRBEnv("workforce5_c++.log");

// Create initial model
GRBModel model = GRBModel(*env);
model.set(GRB_StringAttr_ModelName, "workforce5_c++");

// Initialize assignment decision variables:
//  x[w][s] == 1 if worker w is assigned to shift s.
// This is no longer a pure assignment model, so we must
// use binary variables.
x = new GRBVar*[nWorkers];
for (w = 0; w < nWorkers; w++) {
  x[w] = model.addVars(nShifts, GRB_BINARY);
  xCt++;
  for (s = 0; s < nShifts; s++) {
    ostringstream vname;

    vname << Workers[w] << "." << Shifts[s];
    x[w][s].set(GRB_DoubleAttr_UB, availability[w][s]);
    x[w][s].set(GRB_StringAttr_VarName, vname.str());
  }
}

// Initialize slack decision variables
slacks = model.addVars(nShifts);
for (s = 0; s < nShifts; s++) {
  ostringstream vname;

  vname << Shifts[s] << "Slack";
  slacks[s].set(GRB_StringAttr_VarName, vname.str());
}
```

```
// Variable to represent the total slack
GRBVar totSlack = model.addVar(0, GRB_INFINITY, 0, GRB_CONTINUOUS,
                               "totSlack");

// Initialize variables to count the total shifts worked by each worker
totShifts = model.addVars(nWorkers);
for (w = 0; w < nWorkers; w++) {
  ostringstream vname;

  vname << Workers[w] << "TotShifts";
  totShifts[w].set(GRB_StringAttr_VarName, vname.str());
}

GRBLinExpr lhs;

// Constraint: assign exactly shiftRequirements[s] workers
// to each shift s, plus the slack
for (s = 0; s < nShifts; s++) {
  lhs = 0;
  lhs += slacks[s];
  for (w = 0; w < nWorkers; w++) {
    lhs += x[w][s];
  }
  model.addConstr(lhs == shiftRequirements[s], Shifts[s]);
}

// Constraint: set totSlack column equal to the total slack
lhs = 0;
for (s = 0; s < nShifts; s++) {
  lhs += slacks[s];
}
model.addConstr(lhs == totSlack, "totSlack");

// Constraint: compute the total number of shifts for each worker
for (w = 0; w < nWorkers; w++) {
  lhs = 0;

  for (s = 0; s < nShifts; s++) {
    lhs += x[w][s];
  }

  ostringstream vname;
  vname << "totShifts" << Workers[w];
  model.addConstr(lhs == totShifts[w], vname.str());
```

```
  }

  // Constraint: set minShift/maxShift variable to less <=/>= to the
  // number of shifts among all workers
  GRBVar minShift = model.addVar(0, GRB_INFINITY, 0, GRB_CONTINUOUS,
                                 "minShift");
  GRBVar maxShift = model.addVar(0, GRB_INFINITY, 0, GRB_CONTINUOUS,
                                 "maxShift");
  model.addGenConstrMin(minShift, totShifts, nWorkers, GRB_INFINITY, "minShift");
  model.addGenConstrMax(maxShift, totShifts, nWorkers, -GRB_INFINITY, "maxShift");

  // Set global sense for ALL objectives
  model.set(GRB_IntAttr_ModelSense, GRB_MINIMIZE);

  // Set primary objective
  model.setObjectiveN(totSlack, 0, 2, 1.0, 2.0, 0.1, "TotalSlack");

  // Set secondary objective
  model.setObjectiveN(maxShift - minShift, 1, 1, 1.0, 0, 0, "Fairness");

  // Save problem
  model.write("workforce5_c++.lp");

  // Optimize
  int status = solveAndPrint(model, totSlack, nWorkers, Workers, totShifts);

  // Delete local variables
  if (status != GRB_OPTIMAL)
    return 1;
}
catch (GRBException e){
  cout << "Error code = " << e.getErrorCode() << endl;
  cout << e.getMessage() << endl;
}
catch (...) {
  cout << "Exception during optimization" << endl;
}

for (s = 0; s < xCt; s++)
  delete[] x[s];
delete[] x;
delete[] slacks;
delete[] totShifts;
delete env;
return 0;
```

```
}

int solveAndPrint(GRBModel& model,
                  GRBVar&   totSlack,
                  int       nWorkers,
                  string*   Workers,
                  GRBVar*   totShifts) throw(GRBException)
{
  model.optimize();
  int status = model.get(GRB_IntAttr_Status);

  if ((status == GRB_INF_OR_UNBD) ||
      (status == GRB_INFEASIBLE)  ||
      (status == GRB_UNBOUNDED)     ) {
    cout << "The model cannot be solved " <<
    "because it is infeasible or unbounded" << endl;
    return status;
  }
  if (status != GRB_OPTIMAL) {
    cout << "Optimization was stopped with status " << status << endl;
    return status;
  }

  // Print total slack and the number of shifts worked for each worker
  cout << endl << "Total slack required: " <<
    totSlack.get(GRB_DoubleAttr_X) << endl;
  for (int w = 0; w < nWorkers; ++w) {
    cout << Workers[w] << " worked " <<
    totShifts[w].get(GRB_DoubleAttr_X) << " shifts" << endl;
  }
  cout << endl;

  return status;
}
```

3.3 Java Examples

This section includes source code for all of the Gurobi Java examples. The same source code can be found in the `examples/java` directory of the Gurobi distribution.

Callback.java

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/*
   This example reads a model from a file, sets up a callback that
   monitors optimization progress and implements a custom
   termination strategy, and outputs progress information to the
   screen and to a log file.

   The termination strategy implemented in this callback stops the
   optimization of a MIP model once at least one of the following two
   conditions have been satisfied:
     1) The optimality gap is less than 10%
     2) At least 10000 nodes have been explored, and an integer feasible
        solution has been found.
   Note that termination is normally handled through Gurobi parameters
   (MIPGap, NodeLimit, etc.).  You should only use a callback for
   termination if the available parameters don't capture your desired
   termination criterion.
*/

import gurobi.*;
import java.io.FileWriter;
import java.io.IOException;

public class Callback extends GRBCallback {
  private double     lastiter;
  private double     lastnode;
  private GRBVar[]   vars;
  private FileWriter logfile;

  public Callback(GRBVar[] xvars, FileWriter xlogfile) {
    lastiter = lastnode = -GRB.INFINITY;
    vars = xvars;
    logfile = xlogfile;
  }

  protected void callback() {
    try {
      if (where == GRB.CB_POLLING) {
```

```
  // Ignore polling callback
} else if (where == GRB.CB_PRESOLVE) {
  // Presolve callback
  int cdels = getIntInfo(GRB.CB_PRE_COLDEL);
  int rdels = getIntInfo(GRB.CB_PRE_ROWDEL);
  if (cdels != 0 || rdels != 0) {
    System.out.println(cdels + " columns and " + rdels
        + " rows are removed");
  }
} else if (where == GRB.CB_SIMPLEX) {
  // Simplex callback
  double itcnt = getDoubleInfo(GRB.CB_SPX_ITRCNT);
  if (itcnt - lastiter >= 100) {
    lastiter = itcnt;
    double obj    = getDoubleInfo(GRB.CB_SPX_OBJVAL);
    int    ispert = getIntInfo(GRB.CB_SPX_ISPERT);
    double pinf   = getDoubleInfo(GRB.CB_SPX_PRIMINF);
    double dinf   = getDoubleInfo(GRB.CB_SPX_DUALINF);
    char ch;
    if (ispert == 0)      ch = ' ';
    else if (ispert == 1) ch = 'S';
    else                  ch = 'P';
    System.out.println(itcnt + " " + obj + ch + " "
        + pinf + " " + dinf);
  }
} else if (where == GRB.CB_MIP) {
  // General MIP callback
  double nodecnt = getDoubleInfo(GRB.CB_MIP_NODCNT);
  double objbst  = getDoubleInfo(GRB.CB_MIP_OBJBST);
  double objbnd  = getDoubleInfo(GRB.CB_MIP_OBJBND);
  int    solcnt  = getIntInfo(GRB.CB_MIP_SOLCNT);
  if (nodecnt - lastnode >= 100) {
    lastnode = nodecnt;
    int actnodes = (int) getDoubleInfo(GRB.CB_MIP_NODLFT);
    int itcnt    = (int) getDoubleInfo(GRB.CB_MIP_ITRCNT);
    int cutcnt   = getIntInfo(GRB.CB_MIP_CUTCNT);
    System.out.println(nodecnt + " " + actnodes + " "
        + itcnt + " " + objbst + " " + objbnd + " "
        + solcnt + " " + cutcnt);
  }
  if (Math.abs(objbst - objbnd) < 0.1 * (1.0 + Math.abs(objbst))) {
    System.out.println("Stop early - 10% gap achieved");
    abort();
  }
  if (nodecnt >= 10000 && solcnt > 0) {
```

```
        System.out.println("Stop early - 10000 nodes explored");
        abort();
      }
    } else if (where == GRB.CB_MIPSOL) {
      // MIP solution callback
      int      nodecnt = (int) getDoubleInfo(GRB.CB_MIPSOL_NODCNT);
      double   obj     = getDoubleInfo(GRB.CB_MIPSOL_OBJ);
      int      solcnt  = getIntInfo(GRB.CB_MIPSOL_SOLCNT);
      double[] x       = getSolution(vars);
      System.out.println("**** New solution at node " + nodecnt
          + ", obj " + obj + ", sol " + solcnt
          + ", x[0] = " + x[0] + " ****");
    } else if (where == GRB.CB_MIPNODE) {
      // MIP node callback
      System.out.println("**** New node ****");
      if (getIntInfo(GRB.CB_MIPNODE_STATUS) == GRB.OPTIMAL) {
        double[] x = getNodeRel(vars);
        setSolution(vars, x);
      }
    } else if (where == GRB.CB_BARRIER) {
      // Barrier callback
      int    itcnt   = getIntInfo(GRB.CB_BARRIER_ITRCNT);
      double primobj = getDoubleInfo(GRB.CB_BARRIER_PRIMOBJ);
      double dualobj = getDoubleInfo(GRB.CB_BARRIER_DUALOBJ);
      double priminf = getDoubleInfo(GRB.CB_BARRIER_PRIMINF);
      double dualinf = getDoubleInfo(GRB.CB_BARRIER_DUALINF);
      double cmpl    = getDoubleInfo(GRB.CB_BARRIER_COMPL);
      System.out.println(itcnt + " " + primobj + " " + dualobj + " "
          + priminf + " " + dualinf + " " + cmpl);
    } else if (where == GRB.CB_MESSAGE) {
      // Message callback
      String msg = getStringInfo(GRB.CB_MSG_STRING);
      if (msg != null) logfile.write(msg);
    }
  } catch (GRBException e) {
    System.out.println("Error code: " + e.getErrorCode());
    System.out.println(e.getMessage());
    e.printStackTrace();
  } catch (Exception e) {
    System.out.println("Error during callback");
    e.printStackTrace();
  }
}

public static void main(String[] args) {
```

```
if (args.length < 1) {
  System.out.println("Usage: java Callback filename");
  System.exit(1);
}

FileWriter logfile = null;

try {
  // Create environment
  GRBEnv env = new GRBEnv();

  // Read model from file
  GRBModel model = new GRBModel(env, args[0]);

  // Turn off display and heuristics
  model.set(GRB.IntParam.OutputFlag, 0);
  model.set(GRB.DoubleParam.Heuristics, 0.0);

  // Open log file
  logfile = new FileWriter("cb.log");

  // Create a callback object and associate it with the model
  GRBVar[] vars = model.getVars();
  Callback cb   = new Callback(vars, logfile);

  model.setCallback(cb);

  // Solve model and capture solution information
  model.optimize();

  System.out.println("");
  System.out.println("Optimization complete");
  if (model.get(GRB.IntAttr.SolCount) == 0) {
    System.out.println("No solution found, optimization status = "
        + model.get(GRB.IntAttr.Status));
  } else {
    System.out.println("Solution found, objective = "
        + model.get(GRB.DoubleAttr.ObjVal));

    String[] vnames = model.get(GRB.StringAttr.VarName, vars);
    double[] x      = model.get(GRB.DoubleAttr.X, vars);

    for (int j = 0; j < vars.length; j++) {
      if (x[j] != 0.0) System.out.println(vnames[j] + " " + x[j]);
```

```
        }
      }

      // Dispose of model and environment
      model.dispose();
      env.dispose();

    } catch (GRBException e) {
      System.out.println("Error code: " + e.getErrorCode());
      System.out.println(e.getMessage());
      e.printStackTrace();
    } catch (Exception e) {
      System.out.println("Error during optimization");
      e.printStackTrace();
    } finally {
      // Close log file
      if (logfile != null) {
        try { logfile.close(); } catch (IOException e) {}
      }
    }
  }
}
```

Dense.java

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* This example formulates and solves the following simple QP model:

     minimize    x + y + x^2 + x*y + y^2 + y*z + z^2
     subject to  x + 2 y + 3 z >= 4
                 x +   y       >= 1

   The example illustrates the use of dense matrices to store A and Q
   (and dense vectors for the other relevant data).  We don't recommend
   that you use dense matrices, but this example may be helpful if you
   already have your data in this format.
*/

import gurobi.*;

public class Dense {

  protected static boolean
    dense_optimize(GRBEnv     env,
                   int        rows,
                   int        cols,
                   double[]   c,      // linear portion of objective function
                   double[][] Q,      // quadratic portion of objective function
                   double[][] A,      // constraint matrix
                   char[]     sense,  // constraint senses
                   double[]   rhs,    // RHS vector
                   double[]   lb,     // variable lower bounds
                   double[]   ub,     // variable upper bounds
                   char[]     vtype,  // variable types (continuous, binary, etc.)
                   double[]   solution) {

    boolean success = false;

    try {
      GRBModel model = new GRBModel(env);

      // Add variables to the model

      GRBVar[] vars = model.addVars(lb, ub, null, vtype, null);

      // Populate A matrix

      for (int i = 0; i < rows; i++) {
```

```
      GRBLinExpr expr = new GRBLinExpr();
      for (int j = 0; j < cols; j++)
        if (A[i][j] != 0)
          expr.addTerm(A[i][j], vars[j]);
      model.addConstr(expr, sense[i], rhs[i], "");
    }

    // Populate objective

    GRBQuadExpr obj = new GRBQuadExpr();
    if (Q != null) {
      for (int i = 0; i < cols; i++)
        for (int j = 0; j < cols; j++)
          if (Q[i][j] != 0)
            obj.addTerm(Q[i][j], vars[i], vars[j]);
      for (int j = 0; j < cols; j++)
        if (c[j] != 0)
          obj.addTerm(c[j], vars[j]);
      model.setObjective(obj);
    }

    // Solve model

    model.optimize();

    // Extract solution

    if (model.get(GRB.IntAttr.Status) == GRB.Status.OPTIMAL) {
      success = true;

      for (int j = 0; j < cols; j++)
        solution[j] = vars[j].get(GRB.DoubleAttr.X);
    }

    model.dispose();

  } catch (GRBException e) {
    System.out.println("Error code: " + e.getErrorCode() + ". " +
        e.getMessage());
    e.printStackTrace();
  }

  return success;
}
```

```
  public static void main(String[] args) {
    try {
      GRBEnv env = new GRBEnv();

      double c[] = new double[] {1, 1, 0};
      double Q[][] = new double[][] {{1, 1, 0}, {0, 1, 1}, {0, 0, 1}};
      double A[][] = new double[][] {{1, 2, 3}, {1, 1, 0}};
      char sense[] = new char[] {'>', '>'};
      double rhs[] = new double[] {4, 1};
      double lb[] = new double[] {0, 0, 0};
      boolean success;
      double sol[] = new double[3];


      success = dense_optimize(env, 2, 3, c, Q, A, sense, rhs,
                               lb, null, null, sol);

      if (success) {
        System.out.println("x: " + sol[0] + ", y: " + sol[1] + ", z: " + sol[2]);
      }

      // Dispose of environment
      env.dispose();

    } catch (GRBException e) {
      System.out.println("Error code: " + e.getErrorCode() + ". " +
          e.getMessage());
      e.printStackTrace();
    }
  }
}
```

Diet.java

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* Solve the classic diet model, showing how to add constraints
   to an existing model. */

import gurobi.*;

public class Diet {

  public static void main(String[] args) {
    try {

      // Nutrition guidelines, based on
      // USDA Dietary Guidelines for Americans, 2005
      // http://www.health.gov/DietaryGuidelines/dga2005/
      String Categories[] =
          new String[] { "calories", "protein", "fat", "sodium" };
      int nCategories = Categories.length;
      double minNutrition[] = new double[] { 1800, 91, 0, 0 };
      double maxNutrition[] = new double[] { 2200, GRB.INFINITY, 65, 1779 };

      // Set of foods
      String Foods[] =
          new String[] { "hamburger", "chicken", "hot dog", "fries",
              "macaroni", "pizza", "salad", "milk", "ice cream" };
      int nFoods = Foods.length;
      double cost[] =
          new double[] { 2.49, 2.89, 1.50, 1.89, 2.09, 1.99, 2.49, 0.89,
              1.59 };

      // Nutrition values for the foods
      double nutritionValues[][] = new double[][] {
          { 410, 24, 26, 730 },   // hamburger
          { 420, 32, 10, 1190 },  // chicken
          { 560, 20, 32, 1800 },  // hot dog
          { 380, 4, 19, 270 },    // fries
          { 320, 12, 10, 930 },   // macaroni
          { 320, 15, 12, 820 },   // pizza
          { 320, 31, 12, 1230 },  // salad
          { 100, 8, 2.5, 125 },   // milk
          { 330, 8, 10, 180 }     // ice cream
          };

      // Model
```

```
GRBEnv env = new GRBEnv();
GRBModel model = new GRBModel(env);
model.set(GRB.StringAttr.ModelName, "diet");

// Create decision variables for the nutrition information,
// which we limit via bounds
GRBVar[] nutrition = new GRBVar[nCategories];
for (int i = 0; i < nCategories; ++i) {
  nutrition[i] =
      model.addVar(minNutrition[i], maxNutrition[i], 0, GRB.CONTINUOUS,
                   Categories[i]);
}

// Create decision variables for the foods to buy
GRBVar[] buy = new GRBVar[nFoods];
for (int j = 0; j < nFoods; ++j) {
  buy[j] =
      model.addVar(0, GRB.INFINITY, cost[j], GRB.CONTINUOUS, Foods[j]);
}

// The objective is to minimize the costs
model.set(GRB.IntAttr.ModelSense, GRB.MINIMIZE);

// Nutrition constraints
for (int i = 0; i < nCategories; ++i) {
  GRBLinExpr ntot = new GRBLinExpr();
  for (int j = 0; j < nFoods; ++j) {
    ntot.addTerm(nutritionValues[j][i], buy[j]);
  }
  model.addConstr(ntot, GRB.EQUAL, nutrition[i], Categories[i]);
}

// Solve
model.optimize();
printSolution(model, buy, nutrition);

System.out.println("\nAdding constraint: at most 6 servings of dairy");
GRBLinExpr lhs = new GRBLinExpr();
lhs.addTerm(1.0, buy[7]);
lhs.addTerm(1.0, buy[8]);
model.addConstr(lhs, GRB.LESS_EQUAL, 6.0, "limit_dairy");

// Solve
model.optimize();
printSolution(model, buy, nutrition);
```

```
      // Dispose of model and environment
      model.dispose();
      env.dispose();

    } catch (GRBException e) {
      System.out.println("Error code: " + e.getErrorCode() + ". " +
          e.getMessage());
    }
  }

  private static void printSolution(GRBModel model, GRBVar[] buy,
                                    GRBVar[] nutrition) throws GRBException {
    if (model.get(GRB.IntAttr.Status) == GRB.Status.OPTIMAL) {
      System.out.println("\nCost: " + model.get(GRB.DoubleAttr.ObjVal));
      System.out.println("\nBuy:");
      for (int j = 0; j < buy.length; ++j) {
        if (buy[j].get(GRB.DoubleAttr.X) > 0.0001) {
          System.out.println(buy[j].get(GRB.StringAttr.VarName) + " " +
              buy[j].get(GRB.DoubleAttr.X));
        }
      }
      System.out.println("\nNutrition:");
      for (int i = 0; i < nutrition.length; ++i) {
        System.out.println(nutrition[i].get(GRB.StringAttr.VarName) + " " +
            nutrition[i].get(GRB.DoubleAttr.X));
      }
    } else {
      System.out.println("No solution");
    }
  }
}
```

Facility.java

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* Facility location: a company currently ships its product from 5 plants
   to 4 warehouses. It is considering closing some plants to reduce
   costs. What plant(s) should the company close, in order to minimize
   transportation and fixed costs?

   Based on an example from Frontline Systems:
   http://www.solver.com/disfacility.htm
   Used with permission.
 */

import gurobi.*;

public class Facility {

  public static void main(String[] args) {
    try {

      // Warehouse demand in thousands of units
      double Demand[] = new double[] { 15, 18, 14, 20 };

      // Plant capacity in thousands of units
      double Capacity[] = new double[] { 20, 22, 17, 19, 18 };

      // Fixed costs for each plant
      double FixedCosts[] =
          new double[] { 12000, 15000, 17000, 13000, 16000 };

      // Transportation costs per thousand units
      double TransCosts[][] =
          new double[][] { { 4000, 2000, 3000, 2500, 4500 },
              { 2500, 2600, 3400, 3000, 4000 },
              { 1200, 1800, 2600, 4100, 3000 },
              { 2200, 2600, 3100, 3700, 3200 } };

      // Number of plants and warehouses
      int nPlants = Capacity.length;
      int nWarehouses = Demand.length;

      // Model
      GRBEnv env = new GRBEnv();
      GRBModel model = new GRBModel(env);
      model.set(GRB.StringAttr.ModelName, "facility");
```

```
// Plant open decision variables: open[p] == 1 if plant p is open.
GRBVar[] open = new GRBVar[nPlants];
for (int p = 0; p < nPlants; ++p) {
  open[p] = model.addVar(0, 1, FixedCosts[p], GRB.BINARY, "Open" + p);
}

// Transportation decision variables: how much to transport from
// a plant p to a warehouse w
GRBVar[][] transport = new GRBVar[nWarehouses][nPlants];
for (int w = 0; w < nWarehouses; ++w) {
  for (int p = 0; p < nPlants; ++p) {
    transport[w][p] =
        model.addVar(0, GRB.INFINITY, TransCosts[w][p], GRB.CONTINUOUS,
                     "Trans" + p + "." + w);
  }
}

// The objective is to minimize the total fixed and variable costs
model.set(GRB.IntAttr.ModelSense, GRB.MINIMIZE);

// Production constraints
// Note that the right-hand limit sets the production to zero if
// the plant is closed
for (int p = 0; p < nPlants; ++p) {
  GRBLinExpr ptot = new GRBLinExpr();
  for (int w = 0; w < nWarehouses; ++w) {
    ptot.addTerm(1.0, transport[w][p]);
  }
  GRBLinExpr limit = new GRBLinExpr();
  limit.addTerm(Capacity[p], open[p]);
  model.addConstr(ptot, GRB.LESS_EQUAL, limit, "Capacity" + p);
}

// Demand constraints
for (int w = 0; w < nWarehouses; ++w) {
  GRBLinExpr dtot = new GRBLinExpr();
  for (int p = 0; p < nPlants; ++p) {
    dtot.addTerm(1.0, transport[w][p]);
  }
  model.addConstr(dtot, GRB.EQUAL, Demand[w], "Demand" + w);
}

// Guess at the starting point: close the plant with the highest
// fixed costs; open all others
```

```
// First, open all plants
for (int p = 0; p < nPlants; ++p) {
  open[p].set(GRB.DoubleAttr.Start, 1.0);
}

// Now close the plant with the highest fixed cost
System.out.println("Initial guess:");
double maxFixed = -GRB.INFINITY;
for (int p = 0; p < nPlants; ++p) {
  if (FixedCosts[p] > maxFixed) {
    maxFixed = FixedCosts[p];
  }
}
for (int p = 0; p < nPlants; ++p) {
  if (FixedCosts[p] == maxFixed) {
    open[p].set(GRB.DoubleAttr.Start, 0.0);
    System.out.println("Closing plant " + p + "\n");
    break;
  }
}

// Use barrier to solve root relaxation
model.set(GRB.IntParam.Method, GRB.METHOD_BARRIER);

// Solve
model.optimize();

// Print solution
System.out.println("\nTOTAL COSTS: " + model.get(GRB.DoubleAttr.ObjVal));
System.out.println("SOLUTION:");
for (int p = 0; p < nPlants; ++p) {
  if (open[p].get(GRB.DoubleAttr.X) > 0.99) {
    System.out.println("Plant " + p + " open:");
    for (int w = 0; w < nWarehouses; ++w) {
      if (transport[w][p].get(GRB.DoubleAttr.X) > 0.0001) {
        System.out.println("  Transport " +
            transport[w][p].get(GRB.DoubleAttr.X) +
            " units to warehouse " + w);
      }
    }
  } else {
    System.out.println("Plant " + p + " closed!");
  }
}
```

```
      // Dispose of model and environment
      model.dispose();
      env.dispose();

    } catch (GRBException e) {
      System.out.println("Error code: " + e.getErrorCode() + ". " +
          e.getMessage());
    }
  }
}
```

Feasopt.java

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* This example reads a MIP model from a file, adds artificial
   variables to each constraint, and then minimizes the sum of the
   artificial variables.  A solution with objective zero corresponds
   to a feasible solution to the input model.
   We can also use FeasRelax feature to do it. In this example, we
   use minrelax=1, i.e. optimizing the returned model finds a solution
   that minimizes the original objective, but only from among those
   solutions that minimize the sum of the artificial variables. */

import gurobi.*;

public class Feasopt {
  public static void main(String[] args) {

    if (args.length < 1) {
      System.out.println("Usage: java Feasopt filename");
      System.exit(1);
    }

    try {
      GRBEnv env = new GRBEnv();
      GRBModel feasmodel = new GRBModel(env, args[0]);

      // Create a copy to use FeasRelax feature later */
      GRBModel feasmodel1 = new GRBModel(feasmodel);

      // Clear objective
      feasmodel.setObjective(new GRBLinExpr());

      // Add slack variables
      GRBConstr[] c = feasmodel.getConstrs();
      for (int i = 0; i < c.length; ++i) {
        char sense = c[i].get(GRB.CharAttr.Sense);
        if (sense != '>') {
          GRBConstr[] constrs = new GRBConstr[] { c[i] };
          double[] coeffs = new double[] { -1 };
          feasmodel.addVar(0.0, GRB.INFINITY, 1.0, GRB.CONTINUOUS, constrs,
                           coeffs, "ArtN_" +
                               c[i].get(GRB.StringAttr.ConstrName));
        }
        if (sense != '<') {
          GRBConstr[] constrs = new GRBConstr[] { c[i] };
```

```
          double[] coeffs = new double[] { 1 };
          feasmodel.addVar(0.0, GRB.INFINITY, 1.0, GRB.CONTINUOUS, constrs,
                           coeffs, "ArtP_" +
                               c[i].get(GRB.StringAttr.ConstrName));
        }
      }

      // Optimize modified model
      feasmodel.optimize();
      feasmodel.write("feasopt.lp");

      // use FeasRelax feature */
      feasmodel1.feasRelax(GRB.FEASRELAX_LINEAR, true, false, true);
      feasmodel1.write("feasopt1.lp");
      feasmodel1.optimize();

      // Dispose of model and environment
      feasmodel1.dispose();
      feasmodel.dispose();
      env.dispose();

    } catch (GRBException e) {
      System.out.println("Error code: " + e.getErrorCode() + ". " +
          e.getMessage());
    }
  }
}
```

Fixanddive.java

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* Implement a simple MIP heuristic.  Relax the model,
 sort variables based on fractionality, and fix the 25% of
 the fractional variables that are closest to integer variables.
 Repeat until either the relaxation is integer feasible or
 linearly infeasible. */

import gurobi.*;
import java.util.*;

public class Fixanddive {
  public static void main(String[] args) {

    // Comparison class used to sort variable list based on relaxation
    // fractionality

    class FractionalCompare implements Comparator<GRBVar> {
      public int compare(GRBVar v1, GRBVar v2) {
        try {
          double sol1 = Math.abs(v1.get(GRB.DoubleAttr.X));
          double sol2 = Math.abs(v2.get(GRB.DoubleAttr.X));
          double frac1 = Math.abs(sol1 - Math.floor(sol1 + 0.5));
          double frac2 = Math.abs(sol2 - Math.floor(sol2 + 0.5));
          if (frac1 < frac2) {
            return -1;
          } else if (frac1 == frac2) {
            return 0;
          } else {
            return 1;
          }
        } catch (GRBException e) {
          System.out.println("Error code: " + e.getErrorCode() + ". " +
              e.getMessage());
        }
        return 0;
      }
    }

    if (args.length < 1) {
      System.out.println("Usage: java Fixanddive filename");
      System.exit(1);
    }
```

```
try {
  // Read model
  GRBEnv env = new GRBEnv();
  GRBModel model = new GRBModel(env, args[0]);

  // Collect integer variables and relax them
  ArrayList<GRBVar> intvars = new ArrayList<GRBVar>();
  for (GRBVar v : model.getVars()) {
    if (v.get(GRB.CharAttr.VType) != GRB.CONTINUOUS) {
      intvars.add(v);
      v.set(GRB.CharAttr.VType, GRB.CONTINUOUS);
    }
  }

  model.set(GRB.IntParam.OutputFlag, 0);
  model.optimize();

  // Perform multiple iterations. In each iteration, identify the first
  // quartile of integer variables that are closest to an integer value
  // in the relaxation, fix them to the nearest integer, and repeat.

  for (int iter = 0; iter < 1000; ++iter) {

    // create a list of fractional variables, sorted in order of
    // increasing distance from the relaxation solution to the nearest
    // integer value

    ArrayList<GRBVar> fractional = new ArrayList<GRBVar>();
    for (GRBVar v : intvars) {
      double sol = Math.abs(v.get(GRB.DoubleAttr.X));
      if (Math.abs(sol - Math.floor(sol + 0.5)) > 1e-5) {
        fractional.add(v);
      }
    }

    System.out.println("Iteration " + iter + ", obj " +
        model.get(GRB.DoubleAttr.ObjVal) + ", fractional " +
        fractional.size());

    if (fractional.size() == 0) {
      System.out.println("Found feasible solution - objective " +
          model.get(GRB.DoubleAttr.ObjVal));
      break;
    }
```

```
        // Fix the first quartile to the nearest integer value

        Collections.sort(fractional, new FractionalCompare());
        int nfix = Math.max(fractional.size() / 4, 1);
        for (int i = 0; i < nfix; ++i) {
          GRBVar v = fractional.get(i);
          double fixval = Math.floor(v.get(GRB.DoubleAttr.X) + 0.5);
          v.set(GRB.DoubleAttr.LB, fixval);
          v.set(GRB.DoubleAttr.UB, fixval);
          System.out.println("  Fix " + v.get(GRB.StringAttr.VarName) +
              " to " + fixval + " ( rel " + v.get(GRB.DoubleAttr.X) + " )");
        }

        model.optimize();

        // Check optimization result

        if (model.get(GRB.IntAttr.Status) != GRB.Status.OPTIMAL) {
          System.out.println("Relaxation is infeasible");
          break;
        }
      }

      // Dispose of model and environment
      model.dispose();
      env.dispose();

    } catch (GRBException e) {
      System.out.println("Error code: " + e.getErrorCode() + ". " +
          e.getMessage());
    }
  }

}
```

Genconstr.java

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* In this example we show the use of general constraints for modeling
   some common expressions. We use as an example a SAT-problem where we
   want to see if it is possible to satisfy at least four (or all) clauses
   of the logical for

   L = (x0 or ~x1 or x2)  and (x1 or ~x2 or x3)  and
       (x2 or ~x3 or x0)  and (x3 or ~x0 or x1)  and
       (~x0 or ~x1 or x2) and (~x1 or ~x2 or x3) and
       (~x2 or ~x3 or x0) and (~x3 or ~x0 or x1)

   We do this by introducing two variables for each literal (itself and its
   negated value), a variable for each clause, and then two
   variables for indicating if we can satisfy four, and another to identify
   the minimum of the clauses (so if it one, we can satisfy all clauses)
   and put these two variables in the objective.
   i.e. the Objective function will be

   maximize Obj0 + Obj1

    Obj0 = MIN(Clause1, ... , Clause8)
    Obj1 = 1 -> Clause1 + ... + Clause8 >= 4

   thus, the objective value will be two if and only if we can satisfy all
   clauses; one if and only if at least four clauses can be satisfied, and
   zero otherwise.
 */


import gurobi.*;

public class Genconstr {

  public static final int n = 4;
  public static final int NLITERALS = 4;  // same as n
  public static final int NCLAUSES = 8;
  public static final int NOBJ = 2;

  public static void main(String[] args) {

    try {
      // Example data:
      //   e.g. {0, n+1, 2} means clause (x0 or ~x1 or x2)
```

```
int Clauses[][] = new int[][]
                  {{  0, n+1, 2}, {  1, n+2, 3},
                   {  2, n+3, 0}, {  3, n+0, 1},
                   {n+0, n+1, 2}, {n+1, n+2, 3},
                   {n+2, n+3, 0}, {n+3, n+0, 1}};

int i, status, nSolutions;

// Create environment
GRBEnv env = new GRBEnv("Genconstr.log");

// Create initial model
GRBModel model = new GRBModel(env);
model.set(GRB.StringAttr.ModelName, "Genconstr");

// Initialize decision variables and objective

GRBVar[] Lit     = new GRBVar[NLITERALS];
GRBVar[] NotLit  = new GRBVar[NLITERALS];
for (i = 0; i < NLITERALS; i++) {
  Lit[i]    = model.addVar(0.0, 1.0, 0.0, GRB.BINARY, "X" + String.valueOf(i));
  NotLit[i] = model.addVar(0.0, 1.0, 0.0, GRB.BINARY, "notX" + String.valueOf(i));
}

GRBVar[] Cla = new GRBVar[NCLAUSES];
for (i = 0; i < NCLAUSES; i++) {
  Cla[i] = model.addVar(0.0, 1.0, 0.0, GRB.BINARY, "Clause" + String.valueOf(i));
}

GRBVar[] Obj = new GRBVar[NOBJ];
for (i = 0; i < NOBJ; i++) {
  Obj[i] = model.addVar(0.0, 1.0, 1.0, GRB.BINARY, "Obj" + String.valueOf(i));
}

// Link Xi and notXi
GRBLinExpr lhs;
for (i = 0; i < NLITERALS; i++) {
  lhs = new GRBLinExpr();
  lhs.addTerm(1.0, Lit[i]);
  lhs.addTerm(1.0, NotLit[i]);
  model.addConstr(lhs, GRB.EQUAL, 1.0, "CNSTR_X" + String.valueOf(i));
}

// Link clauses and literals
for (i = 0; i < NCLAUSES; i++) {
```

```
  GRBVar[] clause = new GRBVar[3];
  for (int j = 0; j < 3; j++) {
    if (Clauses[i][j] >= n) clause[j] = NotLit[Clauses[i][j]-n];
    else                    clause[j] = Lit[Clauses[i][j]];
  }
  model.addGenConstrOr(Cla[i], clause, "CNSTR_Clause" + String.valueOf(i));
}

// Link objs with clauses
model.addGenConstrMin(Obj[0], Cla, GRB.INFINITY, "CNSTR_Obj0");
lhs = new GRBLinExpr();
for (i = 0; i < NCLAUSES; i++) {
  lhs.addTerm(1.0, Cla[i]);
}
model.addGenConstrIndicator(Obj[1], 1, lhs, GRB.GREATER_EQUAL, 4.0, "CNSTR_Obj1");

// Set global objective sense
model.set(GRB.IntAttr.ModelSense, GRB.MAXIMIZE);

// Save problem
model.write("Genconstr.mps");
model.write("Genconstr.lp");

// Optimize
model.optimize();

// Status checking
status = model.get(GRB.IntAttr.Status);

if (status == GRB.INF_OR_UNBD ||
    status == GRB.INFEASIBLE  ||
    status == GRB.UNBOUNDED     ) {
  System.out.println("The model cannot be solved " +
        "because it is infeasible or unbounded");
  System.exit(1);
}
if (status != GRB.OPTIMAL) {
  System.out.println("Optimization was stopped with status " + status);
  System.exit(1);
}

// Print result
double objval = model.get(GRB.DoubleAttr.ObjVal);

if (objval > 1.9)
```

```
        System.out.println("Logical expression is satisfiable");
      else if (objval > 0.9)
        System.out.println("At least four clauses can be satisfied");
      else
        System.out.println("Not even three clauses can be satisfied");

      // Dispose of model and environment
      model.dispose();
      env.dispose();

    } catch (GRBException e) {
      System.out.println("Error code: " + e.getErrorCode() + ". " +
          e.getMessage());
    }
  }
}
```

Lp.java

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* This example reads an LP model from a file and solves it.
   If the model is infeasible or unbounded, the example turns off
   presolve and solves the model again. If the model is infeasible,
   the example computes an Irreducible Inconsistent Subsystem (IIS),
   and writes it to a file */

import gurobi.*;

public class Lp {
  public static void main(String[] args) {

    if (args.length < 1) {
      System.out.println("Usage: java Lp filename");
      System.exit(1);
    }

    try {
      GRBEnv env = new GRBEnv();
      GRBModel model = new GRBModel(env, args[0]);

      model.optimize();

      int optimstatus = model.get(GRB.IntAttr.Status);

      if (optimstatus == GRB.Status.INF_OR_UNBD) {
        model.set(GRB.IntParam.Presolve, 0);
        model.optimize();
        optimstatus = model.get(GRB.IntAttr.Status);
      }

      if (optimstatus == GRB.Status.OPTIMAL) {
        double objval = model.get(GRB.DoubleAttr.ObjVal);
        System.out.println("Optimal objective: " + objval);
      } else if (optimstatus == GRB.Status.INFEASIBLE) {
        System.out.println("Model is infeasible");

        // Compute and write out IIS
        model.computeIIS();
        model.write("model.ilp");
      } else if (optimstatus == GRB.Status.UNBOUNDED) {
        System.out.println("Model is unbounded");
      } else {
```

```
        System.out.println("Optimization was stopped with status = "
                           + optimstatus);
      }

      // Dispose of model and environment
      model.dispose();
      env.dispose();

    } catch (GRBException e) {
      System.out.println("Error code: " + e.getErrorCode() + ". " +
          e.getMessage());
    }
  }
}
```

Lpmethod.java

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* Solve a model with different values of the Method parameter;
   show which value gives the shortest solve time. */

import gurobi.*;

public class Lpmethod {

  public static void main(String[] args) {

    if (args.length < 1) {
      System.out.println("Usage: java Lpmethod filename");
      System.exit(1);
    }

    try {
      // Read model
      GRBEnv env = new GRBEnv();
      GRBModel model = new GRBModel(env, args[0]);

      // Solve the model with different values of Method
      int bestMethod = -1;
      double bestTime = model.get(GRB.DoubleParam.TimeLimit);
      for (int i = 0; i <= 2; ++i) {
        model.reset();
        model.set(GRB.IntParam.Method, i);
        model.optimize();
        if (model.get(GRB.IntAttr.Status) == GRB.Status.OPTIMAL) {
          bestTime = model.get(GRB.DoubleAttr.Runtime);
          bestMethod = i;
          // Reduce the TimeLimit parameter to save time
          // with other methods
          model.set(GRB.DoubleParam.TimeLimit, bestTime);
        }
      }

      // Report which method was fastest
      if (bestMethod == -1) {
        System.out.println("Unable to solve this model");
      } else {
        System.out.println("Solved in " + bestTime
            + " seconds with Method: " + bestMethod);
      }
```

```
      // Dispose of model and environment
      model.dispose();
      env.dispose();

    } catch (GRBException e) {
      System.out.println("Error code: " + e.getErrorCode() + ". "
          + e.getMessage());
    }
  }
}
```

Lpmod.java

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* This example reads an LP model from a file and solves it.
   If the model can be solved, then it finds the smallest positive variable,
   sets its upper bound to zero, and resolves the model two ways:
   first with an advanced start, then without an advanced start
   (i.e. 'from scratch'). */

import gurobi.*;

public class Lpmod {
  public static void main(String[] args) {

    if (args.length < 1) {
      System.out.println("Usage: java Lpmod filename");
      System.exit(1);
    }

    try {
      // Read model and determine whether it is an LP
      GRBEnv env = new GRBEnv();
      GRBModel model = new GRBModel(env, args[0]);
      if (model.get(GRB.IntAttr.IsMIP) != 0) {
        System.out.println("The model is not a linear program");
        System.exit(1);
      }

      model.optimize();

      int status = model.get(GRB.IntAttr.Status);

      if (status == GRB.Status.INF_OR_UNBD ||
          status == GRB.Status.INFEASIBLE  ||
          status == GRB.Status.UNBOUNDED     ) {
        System.out.println("The model cannot be solved because it is "
            + "infeasible or unbounded");
        System.exit(1);
      }

      if (status != GRB.Status.OPTIMAL) {
        System.out.println("Optimization was stopped with status " + status);
        System.exit(0);
      }
```

```
    // Find the smallest variable value
    double minVal = GRB.INFINITY;
    GRBVar minVar = null;
    for (GRBVar v : model.getVars()) {
      double sol = v.get(GRB.DoubleAttr.X);
      if ((sol > 0.0001) && (sol < minVal) &&
          (v.get(GRB.DoubleAttr.LB) == 0.0)) {
        minVal = sol;
        minVar = v;
      }
    }

    System.out.println("\n*** Setting " +
        minVar.get(GRB.StringAttr.VarName) + " from " + minVal +
        " to zero ***\n");
    minVar.set(GRB.DoubleAttr.UB, 0.0);

    // Solve from this starting point
    model.optimize();

    // Save iteration & time info
    double warmCount = model.get(GRB.DoubleAttr.IterCount);
    double warmTime = model.get(GRB.DoubleAttr.Runtime);

    // Reset the model and resolve
    System.out.println("\n*** Resetting and solving "
        + "without an advanced start ***\n");
    model.reset();
    model.optimize();

    double coldCount = model.get(GRB.DoubleAttr.IterCount);
    double coldTime = model.get(GRB.DoubleAttr.Runtime);

    System.out.println("\n*** Warm start: " + warmCount + " iterations, " +
        warmTime + " seconds");
    System.out.println("*** Cold start: " + coldCount + " iterations, " +
        coldTime + " seconds");

    // Dispose of model and environment
    model.dispose();
    env.dispose();

  } catch (GRBException e) {
    System.out.println("Error code: " + e.getErrorCode() + ". " +
        e.getMessage());
```

```
    }
  }
}
```

Mip1.java

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* This example formulates and solves the following simple MIP model:

     maximize    x +   y + 2 z
     subject to  x + 2 y + 3 z <= 4
                 x +   y       >= 1
     x, y, z binary
*/

import gurobi.*;

public class Mip1 {
  public static void main(String[] args) {
    try {
      GRBEnv    env   = new GRBEnv("mip1.log");
      GRBModel  model = new GRBModel(env);

      // Create variables

      GRBVar x = model.addVar(0.0, 1.0, 0.0, GRB.BINARY, "x");
      GRBVar y = model.addVar(0.0, 1.0, 0.0, GRB.BINARY, "y");
      GRBVar z = model.addVar(0.0, 1.0, 0.0, GRB.BINARY, "z");

      // Set objective: maximize x + y + 2 z

      GRBLinExpr expr = new GRBLinExpr();
      expr.addTerm(1.0, x); expr.addTerm(1.0, y); expr.addTerm(2.0, z);
      model.setObjective(expr, GRB.MAXIMIZE);

      // Add constraint: x + 2 y + 3 z <= 4

      expr = new GRBLinExpr();
      expr.addTerm(1.0, x); expr.addTerm(2.0, y); expr.addTerm(3.0, z);
      model.addConstr(expr, GRB.LESS_EQUAL, 4.0, "c0");

      // Add constraint: x + y >= 1

      expr = new GRBLinExpr();
      expr.addTerm(1.0, x); expr.addTerm(1.0, y);
      model.addConstr(expr, GRB.GREATER_EQUAL, 1.0, "c1");

      // Optimize model
```

```
      model.optimize();

      System.out.println(x.get(GRB.StringAttr.VarName)
                         + " " +x.get(GRB.DoubleAttr.X));
      System.out.println(y.get(GRB.StringAttr.VarName)
                         + " " +y.get(GRB.DoubleAttr.X));
      System.out.println(z.get(GRB.StringAttr.VarName)
                         + " " +z.get(GRB.DoubleAttr.X));

      System.out.println("Obj: " + model.get(GRB.DoubleAttr.ObjVal));

      // Dispose of model and environment

      model.dispose();
      env.dispose();

    } catch (GRBException e) {
      System.out.println("Error code: " + e.getErrorCode() + ". " +
                         e.getMessage());
    }
  }
}
```

Mip2.java

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* This example reads a MIP model from a file, solves it and
   prints the objective values from all feasible solutions
   generated while solving the MIP. Then it creates the fixed
   model and solves that model. */

import gurobi.*;

public class Mip2 {
  public static void main(String[] args) {

    if (args.length < 1) {
      System.out.println("Usage: java Mip2 filename");
      System.exit(1);
    }

    try {
      GRBEnv env = new GRBEnv();
      GRBModel model = new GRBModel(env, args[0]);
      if (model.get(GRB.IntAttr.IsMIP) == 0) {
        System.out.println("Model is not a MIP");
        System.exit(1);
      }

      model.optimize();

      int optimstatus = model.get(GRB.IntAttr.Status);
      double objval = 0;
      if (optimstatus == GRB.Status.OPTIMAL) {
        objval = model.get(GRB.DoubleAttr.ObjVal);
        System.out.println("Optimal objective: " + objval);
      } else if (optimstatus == GRB.Status.INF_OR_UNBD) {
        System.out.println("Model is infeasible or unbounded");
        return;
      } else if (optimstatus == GRB.Status.INFEASIBLE) {
        System.out.println("Model is infeasible");
        return;
      } else if (optimstatus == GRB.Status.UNBOUNDED) {
        System.out.println("Model is unbounded");
        return;
      } else {
        System.out.println("Optimization was stopped with status = "
            + optimstatus);
```

```
  return;
}

/* Iterate over the solutions and compute the objectives */
GRBVar[] vars = model.getVars();
model.set(GRB.IntParam.OutputFlag, 0);

System.out.println();
for (int k = 0; k < model.get(GRB.IntAttr.SolCount); ++k) {
  model.set(GRB.IntParam.SolutionNumber, k);
  double objn = 0.0;

  for (int j = 0; j < vars.length; j++) {
    objn += vars[j].get(GRB.DoubleAttr.Obj)
        * vars[j].get(GRB.DoubleAttr.Xn);
  }

  System.out.println("Solution " + k + " has objective: " + objn);
}
System.out.println();
model.set(GRB.IntParam.OutputFlag, 1);

/* Create a fixed model, turn off presolve and solve */

GRBModel fixed = model.fixedModel();

fixed.set(GRB.IntParam.Presolve, 0);

fixed.optimize();

int foptimstatus = fixed.get(GRB.IntAttr.Status);

if (foptimstatus != GRB.Status.OPTIMAL) {
  System.err.println("Error: fixed model isn't optimal");
  return;
}

double fobjval = fixed.get(GRB.DoubleAttr.ObjVal);

if (Math.abs(fobjval - objval) > 1.0e-6 * (1.0 + Math.abs(objval))) {
  System.err.println("Error: objective values are different");
  return;
}

GRBVar[] fvars  = fixed.getVars();
```

```
      double[] x      = fixed.get(GRB.DoubleAttr.X, fvars);
      String[] vnames = fixed.get(GRB.StringAttr.VarName, fvars);

      for (int j = 0; j < fvars.length; j++) {
        if (x[j] != 0.0) {
          System.out.println(vnames[j] + " " + x[j]);
        }
      }

      // Dispose of models and environment
      fixed.dispose();
      model.dispose();
      env.dispose();

    } catch (GRBException e) {
      System.out.println("Error code: " + e.getErrorCode() + ". "
          + e.getMessage());
    }
  }
}
```

Multiobj.java

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* Want to cover three different sets but subject to a common budget of
   elements allowed to be used. However, the sets have different priorities to
   be covered; and we tackle this by using multi-objective optimization. */

import gurobi.*;

public class Multiobj {
  public static void main(String[] args) {

    try {
      // Sample data
      int groundSetSize = 20;
      int nSubsets      = 4;
      int Budget        = 12;
      double Set[][] = new double[][]
      { { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
        { 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1 },
        { 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0 },
        { 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0 } };
      int    SetObjPriority[] = new int[] {3, 2, 2, 1};
      double SetObjWeight[]   = new double[] {1.0, 0.25, 1.25, 1.0};
      int e, i, status, nSolutions;

      // Create environment
      GRBEnv env = new GRBEnv("Multiobj.log");

      // Create initial model
      GRBModel model = new GRBModel(env);
      model.set(GRB.StringAttr.ModelName, "Multiobj");

      // Initialize decision variables for ground set:
      // x[e] == 1 if element e is chosen for the covering.
      GRBVar[] Elem = model.addVars(groundSetSize, GRB.BINARY);
      for (e = 0; e < groundSetSize; e++) {
        String vname = "El" + String.valueOf(e);
        Elem[e].set(GRB.StringAttr.VarName, vname);
      }

      // Constraint: limit total number of elements to be picked to be at most
      // Budget
      GRBLinExpr lhs = new GRBLinExpr();
      for (e = 0; e < groundSetSize; e++) {
```

```
  lhs.addTerm(1.0, Elem[e]);
}
model.addConstr(lhs, GRB.LESS_EQUAL, Budget, "Budget");

// Set global sense for ALL objectives
model.set(GRB.IntAttr.ModelSense, GRB.MAXIMIZE);

// Limit how many solutions to collect
model.set(GRB.IntParam.PoolSolutions, 100);

// Set and configure i-th objective
for (i = 0; i < nSubsets; i++) {
  GRBLinExpr objn = new GRBLinExpr();
  String vname = "Set" + String.valueOf(i);

  for (e = 0; e < groundSetSize; e++)
    objn.addTerm(Set[i][e], Elem[e]);

  model.setObjectiveN(objn, i, SetObjPriority[i], SetObjWeight[i],
                      1.0 + i, 0.01, vname);
}

// Save problem
model.write("Multiobj.lp");

// Optimize
model.optimize();

// Status checking
status = model.get(GRB.IntAttr.Status);

if (status == GRB.INF_OR_UNBD ||
    status == GRB.INFEASIBLE  ||
    status == GRB.UNBOUNDED     ) {
  System.out.println("The model cannot be solved " +
           "because it is infeasible or unbounded");
  System.exit(1);
}
if (status != GRB.OPTIMAL) {
  System.out.println("Optimization was stopped with status " + status);
  System.exit(1);
}

// Print best selected set
System.out.println("Selected elements in best solution:");
```

```
      System.out.println("\t");
      for (e = 0; e < groundSetSize; e++) {
        if (Elem[e].get(GRB.DoubleAttr.X) < .9) continue;
        System.out.print(" El" + e);
      }
      System.out.println();

      // Print number of solutions stored
      nSolutions = model.get(GRB.IntAttr.SolCount);
      System.out.println("Number of solutions found: " + nSolutions);

      // Print objective values of solutions
      if (nSolutions > 10) nSolutions = 10;
      System.out.println("Objective values for first " + nSolutions);
      System.out.println(" solutions:");
      for (i = 0; i < nSubsets; i++) {
        model.set(GRB.IntParam.ObjNumber, i);

        System.out.print("\tSet" + i);
        for (e = 0; e < nSolutions; e++) {
          System.out.print(" ");
          model.set(GRB.IntParam.SolutionNumber, e);
          double val = model.get(GRB.DoubleAttr.ObjNVal);
          System.out.print("      " + val);
        }
        System.out.println();
      }
      model.dispose();
      env.dispose();
    } catch (GRBException e) {
      System.out.println("Error code = " + e.getErrorCode());
      System.out.println(e.getMessage());
    }
  }
}
```

Params.java

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* Use parameters that are associated with a model.

   A MIP is solved for a few seconds with different sets of parameters.
   The one with the smallest MIP gap is selected, and the optimization
   is resumed until the optimal solution is found.
*/

import gurobi.*;

public class Params {

  public static void main(String[] args) {

    if (args.length < 1) {
      System.out.println("Usage: java Params filename");
      System.exit(1);
    }

    try {
      // Read model and verify that it is a MIP
      GRBEnv env = new GRBEnv();
      GRBModel m = new GRBModel(env, args[0]);
      if (m.get(GRB.IntAttr.IsMIP) == 0) {
        System.out.println("The model is not an integer program");
        System.exit(1);
      }

      // Set a 2 second time limit
      m.set(GRB.DoubleParam.TimeLimit, 2);

      // Now solve the model with different values of MIPFocus
      GRBModel bestModel = new GRBModel(m);
      bestModel.optimize();
      for (int i = 1; i <= 3; ++i) {
        m.reset();
        m.set(GRB.IntParam.MIPFocus, i);
        m.optimize();
        if (bestModel.get(GRB.DoubleAttr.MIPGap) >
                    m.get(GRB.DoubleAttr.MIPGap)) {
          GRBModel swap = bestModel;
          bestModel = m;
          m = swap;
```

```
        }
      }

      // Finally, delete the extra model, reset the time limit and
      // continue to solve the best model to optimality
      m.dispose();
      bestModel.set(GRB.DoubleParam.TimeLimit, GRB.INFINITY);
      bestModel.optimize();
      System.out.println("Solved with MIPFocus: " +
          bestModel.get(GRB.IntParam.MIPFocus));

    } catch (GRBException e) {
      System.out.println("Error code: " + e.getErrorCode() + ". " +
          e.getMessage());
    }
  }
}
```

Piecewise.java

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* This example considers the following separable, convex problem:

     minimize    f(x) - y + g(z)
     subject to  x + 2 y + 3 z <= 4
                 x +   y       >= 1
                 x,    y,    z <= 1

   where f(u) = exp(-u) and g(u) = 2 u^2 - 4 u, for all real u. It
   formulates and solves a simpler LP model by approximating f and
   g with piecewise-linear functions. Then it transforms the model
   into a MIP by negating the approximation for f, which corresponds
   to a non-convex piecewise-linear function, and solves it again.
*/

import gurobi.*;

public class Piecewise {

  private static double f(double u) { return Math.exp(-u); }
  private static double g(double u) { return 2 * u * u - 4 * u; }

  public static void main(String[] args) {
    try {

      // Create environment

      GRBEnv env = new GRBEnv();

      // Create a new model

      GRBModel model = new GRBModel(env);

      // Create variables

      double lb = 0.0, ub = 1.0;

      GRBVar x = model.addVar(lb, ub, 0.0, GRB.CONTINUOUS, "x");
      GRBVar y = model.addVar(lb, ub, 0.0, GRB.CONTINUOUS, "y");
      GRBVar z = model.addVar(lb, ub, 0.0, GRB.CONTINUOUS, "z");

      // Set objective for y
```

```
GRBLinExpr obj = new GRBLinExpr();
obj.addTerm(-1.0, y);
model.setObjective(obj);

// Add piecewise-linear objective functions for x and z

int npts = 101;
double[] ptu = new double[npts];
double[] ptf = new double[npts];
double[] ptg = new double[npts];

for (int i = 0; i < npts; i++) {
  ptu[i] = lb + (ub - lb) * i / (npts - 1);
  ptf[i] = f(ptu[i]);
  ptg[i] = g(ptu[i]);
}

model.setPWLObj(x, ptu, ptf);
model.setPWLObj(z, ptu, ptg);

// Add constraint: x + 2 y + 3 z <= 4

GRBLinExpr expr = new GRBLinExpr();
expr.addTerm(1.0, x); expr.addTerm(2.0, y); expr.addTerm(3.0, z);
model.addConstr(expr, GRB.LESS_EQUAL, 4.0, "c0");

// Add constraint: x + y >= 1

expr = new GRBLinExpr();
expr.addTerm(1.0, x); expr.addTerm(1.0, y);
model.addConstr(expr, GRB.GREATER_EQUAL, 1.0, "c1");

// Optimize model as an LP

model.optimize();

System.out.println("IsMIP: " + model.get(GRB.IntAttr.IsMIP));

System.out.println(x.get(GRB.StringAttr.VarName)
                   + " " +x.get(GRB.DoubleAttr.X));
System.out.println(y.get(GRB.StringAttr.VarName)
                   + " " +y.get(GRB.DoubleAttr.X));
System.out.println(z.get(GRB.StringAttr.VarName)
                   + " " +z.get(GRB.DoubleAttr.X));
```

```
      System.out.println("Obj: " + model.get(GRB.DoubleAttr.ObjVal));

      System.out.println();

      // Negate piecewise-linear objective function for x

      for (int i = 0; i < npts; i++) {
        ptf[i] = -ptf[i];
      }

      model.setPWLObj(x, ptu, ptf);

      // Optimize model as a MIP

      model.optimize();

      System.out.println("IsMIP: " + model.get(GRB.IntAttr.IsMIP));

      System.out.println(x.get(GRB.StringAttr.VarName)
                         + " " +x.get(GRB.DoubleAttr.X));
      System.out.println(y.get(GRB.StringAttr.VarName)
                         + " " +y.get(GRB.DoubleAttr.X));
      System.out.println(z.get(GRB.StringAttr.VarName)
                         + " " +z.get(GRB.DoubleAttr.X));

      System.out.println("Obj: " + model.get(GRB.DoubleAttr.ObjVal));

      // Dispose of model and environment

      model.dispose();
      env.dispose();

    } catch (GRBException e) {
      System.out.println("Error code: " + e.getErrorCode() + ". " +
          e.getMessage());
    }
  }
}
```

Poolsearch.java

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* We find alternative epsilon-optimal solutions to a given knapsack
   problem by using PoolSearchMode */

import gurobi.*;

public class Poolsearch {

  public static void main(String[] args) {

    try{
      // Sample data
      int groundSetSize = 10;
      double objCoef[] = new double[] {32, 32, 15, 15, 6, 6, 1, 1, 1, 1};
      double knapsackCoef[] = new double[] {16, 16,  8,  8, 4, 4, 2, 2, 1, 1};
      double Budget = 33;
      int e, status, nSolutions;

      // Create environment
      GRBEnv env = new GRBEnv("Poolsearch.log");

      // Create initial model
      GRBModel model = new GRBModel(env);
      model.set(GRB.StringAttr.ModelName, "Poolsearch");

      // Initialize decision variables for ground set:
      // x[e] == 1 if element e is chosen
      GRBVar[] Elem = model.addVars(groundSetSize, GRB.BINARY);
      model.set(GRB.DoubleAttr.Obj, Elem, objCoef, 0, groundSetSize);

      for (e = 0; e < groundSetSize; e++) {
        Elem[e].set(GRB.StringAttr.VarName, "El" + String.valueOf(e));
      }

      // Constraint: limit total number of elements to be picked to be at most
      // Budget
      GRBLinExpr lhs = new GRBLinExpr();
      for (e = 0; e < groundSetSize; e++) {
        lhs.addTerm(knapsackCoef[e], Elem[e]);
      }
      model.addConstr(lhs, GRB.LESS_EQUAL, Budget, "Budget");

      // set global sense for ALL objectives
```

```
model.set(GRB.IntAttr.ModelSense, GRB.MAXIMIZE);

// Limit how many solutions to collect
model.set(GRB.IntParam.PoolSolutions, 1024);

// Limit the search space by setting a gap for the worst possible solution that will be ac
model.set(GRB.DoubleParam.PoolGap, 0.10);

// do a systematic search for the k-best solutions
model.set(GRB.IntParam.PoolSearchMode, 2);

// save problem
model.write("Poolsearch.lp");

// Optimize
model.optimize();

// Status checking
status = model.get(GRB.IntAttr.Status);

if (status == GRB.INF_OR_UNBD ||
    status == GRB.INFEASIBLE  ||
    status == GRB.UNBOUNDED     ) {
  System.out.println("The model cannot be solved " +
         "because it is infeasible or unbounded");
  System.exit(1);
}
if (status != GRB.OPTIMAL) {
  System.out.println("Optimization was stopped with status " + status);
  System.exit(1);
}

// Print best selected set
System.out.println("Selected elements in best solution:");
System.out.print("\t");
for (e = 0; e < groundSetSize; e++) {
  if (Elem[e].get(GRB.DoubleAttr.X) < .9) continue;
  System.out.print(" El" + e);
}
System.out.println();

// Print number of solutions stored
nSolutions = model.get(GRB.IntAttr.SolCount);
System.out.println("Number of solutions found: " + nSolutions);
```

```
      // Print objective values of solutions
      for (e = 0; e < nSolutions; e++) {
        model.set(GRB.IntParam.SolutionNumber, e);
        System.out.print(model.get(GRB.DoubleAttr.PoolObjVal) +  " ");
        if (e%15 == 14) System.out.println();
      }
      System.out.println();

      // print fourth best set if available
      if (nSolutions >= 4) {
        model.set(GRB.IntParam.SolutionNumber, 3);

        System.out.println("Selected elements in fourth best solution:");
        System.out.print("\t");
        for (e = 0; e < groundSetSize; e++) {
          if (Elem[e].get(GRB.DoubleAttr.Xn) < .9) continue;
          System.out.print(" El" + e);
        }
        System.out.println();
      }

      model.dispose();
      env.dispose();
    } catch (GRBException e) {
      System.out.println("Error code: " + e.getErrorCode() + ". " +
          e.getMessage());
    }
  }

}
```

Qcp.java

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* This example formulates and solves the following simple QCP model:

     maximize    x
     subject to  x + y + z = 1
                 x^2 + y^2 <= z^2 (second-order cone)
                 x^2 <= yz        (rotated second-order cone)
*/

import gurobi.*;

public class Qcp {
  public static void main(String[] args) {
    try {
      GRBEnv    env   = new GRBEnv("qcp.log");
      GRBModel  model = new GRBModel(env);

      // Create variables

      GRBVar x = model.addVar(0.0, GRB.INFINITY, 0.0, GRB.CONTINUOUS, "x");
      GRBVar y = model.addVar(0.0, GRB.INFINITY, 0.0, GRB.CONTINUOUS, "y");
      GRBVar z = model.addVar(0.0, GRB.INFINITY, 0.0, GRB.CONTINUOUS, "z");

      // Set objective

      GRBLinExpr obj = new GRBLinExpr();
      obj.addTerm(1.0, x);
      model.setObjective(obj, GRB.MAXIMIZE);

      // Add linear constraint: x + y + z = 1

      GRBLinExpr expr = new GRBLinExpr();
      expr.addTerm(1.0, x); expr.addTerm(1.0, y); expr.addTerm(1.0, z);
      model.addConstr(expr, GRB.EQUAL, 1.0, "c0");

      // Add second-order cone: x^2 + y^2 <= z^2

      GRBQuadExpr qexpr = new GRBQuadExpr();
      qexpr.addTerm(1.0, x, x);
      qexpr.addTerm(1.0, y, y);
      qexpr.addTerm(-1.0, z, z);
      model.addQConstr(qexpr, GRB.LESS_EQUAL, 0.0, "qc0");
```

```
      // Add rotated cone: x^2 <= yz

      qexpr = new GRBQuadExpr();
      qexpr.addTerm(1.0, x, x);
      qexpr.addTerm(-1.0, y, z);
      model.addQConstr(qexpr, GRB.LESS_EQUAL, 0.0, "qc1");

      // Optimize model

      model.optimize();

      System.out.println(x.get(GRB.StringAttr.VarName)
                         + " " +x.get(GRB.DoubleAttr.X));
      System.out.println(y.get(GRB.StringAttr.VarName)
                         + " " +y.get(GRB.DoubleAttr.X));
      System.out.println(z.get(GRB.StringAttr.VarName)
                         + " " +z.get(GRB.DoubleAttr.X));

      System.out.println("Obj: " + model.get(GRB.DoubleAttr.ObjVal) + " " +
                         obj.getValue());
      System.out.println();

      // Dispose of model and environment

      model.dispose();
      env.dispose();

    } catch (GRBException e) {
      System.out.println("Error code: " + e.getErrorCode() + ". " +
          e.getMessage());
    }
  }
}
```

Qp.java

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* This example formulates and solves the following simple QP model:

     minimize    x^2 + x*y + y^2 + y*z + z^2 + 2 x
     subject to  x + 2 y + 3 z >= 4
                 x +   y       >= 1

   It solves it once as a continuous model, and once as an integer model.
*/

import gurobi.*;

public class Qp {
  public static void main(String[] args) {
    try {
      GRBEnv    env   = new GRBEnv("qp.log");
      GRBModel  model = new GRBModel(env);

      // Create variables

      GRBVar x = model.addVar(0.0, 1.0, 0.0, GRB.CONTINUOUS, "x");
      GRBVar y = model.addVar(0.0, 1.0, 0.0, GRB.CONTINUOUS, "y");
      GRBVar z = model.addVar(0.0, 1.0, 0.0, GRB.CONTINUOUS, "z");

      // Set objective

      GRBQuadExpr obj = new GRBQuadExpr();
      obj.addTerm(1.0, x, x);
      obj.addTerm(1.0, x, y);
      obj.addTerm(1.0, y, y);
      obj.addTerm(1.0, y, z);
      obj.addTerm(1.0, z, z);
      obj.addTerm(2.0, x);
      model.setObjective(obj);

      // Add constraint: x + 2 y + 3 z >= 4

      GRBLinExpr expr = new GRBLinExpr();
      expr.addTerm(1.0, x); expr.addTerm(2.0, y); expr.addTerm(3.0, z);
      model.addConstr(expr, GRB.GREATER_EQUAL, 4.0, "c0");

      // Add constraint: x + y >= 1
```

```
expr = new GRBLinExpr();
expr.addTerm(1.0, x); expr.addTerm(1.0, y);
model.addConstr(expr, GRB.GREATER_EQUAL, 1.0, "c1");

// Optimize model

model.optimize();

System.out.println(x.get(GRB.StringAttr.VarName)
                   + " " +x.get(GRB.DoubleAttr.X));
System.out.println(y.get(GRB.StringAttr.VarName)
                   + " " +y.get(GRB.DoubleAttr.X));
System.out.println(z.get(GRB.StringAttr.VarName)
                   + " " +z.get(GRB.DoubleAttr.X));

System.out.println("Obj: " + model.get(GRB.DoubleAttr.ObjVal) + " " +
                   obj.getValue());
System.out.println();


// Change variable types to integer

x.set(GRB.CharAttr.VType, GRB.INTEGER);
y.set(GRB.CharAttr.VType, GRB.INTEGER);
z.set(GRB.CharAttr.VType, GRB.INTEGER);

// Optimize again

model.optimize();

System.out.println(x.get(GRB.StringAttr.VarName)
                   + " " +x.get(GRB.DoubleAttr.X));
System.out.println(y.get(GRB.StringAttr.VarName)
                   + " " +y.get(GRB.DoubleAttr.X));
System.out.println(z.get(GRB.StringAttr.VarName)
                   + " " +z.get(GRB.DoubleAttr.X));

System.out.println("Obj: " + model.get(GRB.DoubleAttr.ObjVal) + " " +
                   obj.getValue());

// Dispose of model and environment

model.dispose();
env.dispose();
```

```
    } catch (GRBException e) {
      System.out.println("Error code: " + e.getErrorCode() + ". " +
          e.getMessage());
    }
  }
}
```

Sensitivity.java

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* A simple sensitivity analysis example which reads a MIP model
   from a file and solves it. Then each binary variable is set
   to 1-X, where X is its value in the optimal solution, and
   the impact on the objective function value is reported.
*/

import gurobi.*;

public class Sensitivity {

  public static void main(String[] args) {

    if (args.length < 1) {
      System.out.println("Usage: java Sensitivity filename");
      System.exit(1);
    }

    try {

      // Create environment

      GRBEnv env = new GRBEnv();

      // Read and solve model

      GRBModel model = new GRBModel(env, args[0]);

      if (model.get(GRB.IntAttr.IsMIP) == 0) {
        System.out.println("Model is not a MIP");
        System.exit(1);
      }

      model.optimize();

      if (model.get(GRB.IntAttr.Status) != GRB.OPTIMAL) {
        System.out.println("Optimization ended with status "
            + model.get(GRB.IntAttr.Status));
        System.exit(1);
      }

      // Store the optimal solution
```

```
double   origObjVal = model.get(GRB.DoubleAttr.ObjVal);
GRBVar[] vars       = model.getVars();
double[] origX      = model.get(GRB.DoubleAttr.X, vars);

// Disable solver output for subsequent solves

model.set(GRB.IntParam.OutputFlag, 0);

// Iterate through unfixed, binary variables in model

for (int i = 0; i < vars.length; i++) {
  GRBVar v     = vars[i];
  char   vType = v.get(GRB.CharAttr.VType);

  if (v.get(GRB.DoubleAttr.LB) == 0 && v.get(GRB.DoubleAttr.UB) == 1
      && (vType == GRB.BINARY || vType == GRB.INTEGER)) {

    // Set variable to 1-X, where X is its value in optimal solution

    if (origX[i] < 0.5) {
      v.set(GRB.DoubleAttr.LB, 1.0);
      v.set(GRB.DoubleAttr.Start, 1.0);
    } else {
      v.set(GRB.DoubleAttr.UB, 0.0);
      v.set(GRB.DoubleAttr.Start, 0.0);
    }

    // Update MIP start for the other variables

    for (int j = 0; j < vars.length; j++) {
      if (j != i) {
        vars[j].set(GRB.DoubleAttr.Start, origX[j]);
      }
    }

    // Solve for new value and capture sensitivity information

    model.optimize();

    if (model.get(GRB.IntAttr.Status) == GRB.OPTIMAL) {
      System.out.println("Objective sensitivity for variable "
          + v.get(GRB.StringAttr.VarName) + " is "
          + (model.get(GRB.DoubleAttr.ObjVal) - origObjVal));
    } else {
      System.out.println("Objective sensitivity for variable "
```

```
                + v.get(GRB.StringAttr.VarName) + " is infinite");
          }

          // Restore the original variable bounds

          v.set(GRB.DoubleAttr.LB, 0.0);
          v.set(GRB.DoubleAttr.UB, 1.0);
        }
      }

      // Dispose of model and environment

      model.dispose();
      env.dispose();

    } catch (GRBException e) {
      System.out.println("Error code: " + e.getErrorCode());
      System.out.println(e.getMessage());
      e.printStackTrace();
    }
  }
}
```

Sos.java

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* This example creates a very simple Special Ordered Set (SOS) model.
   The model consists of 3 continuous variables, no linear constraints,
   and a pair of SOS constraints of type 1. */

import gurobi.*;

public class Sos {
  public static void main(String[] args) {
    try {
      GRBEnv env = new GRBEnv();

      GRBModel model = new GRBModel(env);

      // Create variables

      double ub[]    = {1, 1, 2};
      double obj[]   = {-2, -1, -1};
      String names[] = {"x0", "x1", "x2"};

      GRBVar[] x = model.addVars(null, ub, obj, null, names);

      // Add first SOS1: x0=0 or x1=0

      GRBVar sosv1[]  = {x[0], x[1]};
      double soswt1[] = {1, 2};

      model.addSOS(sosv1, soswt1, GRB.SOS_TYPE1);

      // Add second SOS1: x0=0 or x2=0

      GRBVar sosv2[]  = {x[0], x[2]};
      double soswt2[] = {1, 2};

      model.addSOS(sosv2, soswt2, GRB.SOS_TYPE1);

      // Optimize model

      model.optimize();

      for (int i = 0; i < 3; i++)
        System.out.println(x[i].get(GRB.StringAttr.VarName) + " "
                           + x[i].get(GRB.DoubleAttr.X));
```

```
      // Dispose of model and environment
      model.dispose();
      env.dispose();

    } catch (GRBException e) {
      System.out.println("Error code: " + e.getErrorCode() + ". " +
          e.getMessage());
    }
  }
}
```

Sudoku.java

```
/* Copyright 2017, Gurobi Optimization, Inc. */
/*
  Sudoku example.

  The Sudoku board is a 9x9 grid, which is further divided into a 3x3 grid
  of 3x3 grids.  Each cell in the grid must take a value from 0 to 9.
  No two grid cells in the same row, column, or 3x3 subgrid may take the
  same value.

  In the MIP formulation, binary variables x[i,j,v] indicate whether
  cell <i,j> takes value 'v'.  The constraints are as follows:
    1. Each cell must take exactly one value (sum_v x[i,j,v] = 1)
    2. Each value is used exactly once per row (sum_i x[i,j,v] = 1)
    3. Each value is used exactly once per column (sum_j x[i,j,v] = 1)
    4. Each value is used exactly once per 3x3 subgrid (sum_grid x[i,j,v] = 1)

  Input datasets for this example can be found in examples/data/sudoku*.
*/

import gurobi.*;
import java.io.*;

public class Sudoku {
  public static void main(String[] args) {
    int n = 9;
    int s = 3;

    if (args.length < 1) {
      System.out.println("Usage: java Sudoku filename");
      System.exit(1);
    }

    try {
      GRBEnv env = new GRBEnv();
      GRBModel model = new GRBModel(env);

      // Create 3-D array of model variables

      GRBVar[][][] vars = new GRBVar[n][n][n];

      for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
          for (int v = 0; v < n; v++) {
            String st = "G_" + String.valueOf(i) + "_" + String.valueOf(j)
```

```
                        + "_" + String.valueOf(v);
        vars[i][j][v] = model.addVar(0.0, 1.0, 0.0, GRB.BINARY, st);
      }
    }
  }

  // Add constraints

  GRBLinExpr expr;

  // Each cell must take one value

  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      expr = new GRBLinExpr();
      expr.addTerms(null, vars[i][j]);
      String st = "V_" + String.valueOf(i) + "_" + String.valueOf(j);
      model.addConstr(expr, GRB.EQUAL, 1.0, st);
    }
  }

  // Each value appears once per row

  for (int i = 0; i < n; i++) {
    for (int v = 0; v < n; v++) {
      expr = new GRBLinExpr();
      for (int j = 0; j < n; j++)
        expr.addTerm(1.0, vars[i][j][v]);
      String st = "R_" + String.valueOf(i) + "_" + String.valueOf(v);
      model.addConstr(expr, GRB.EQUAL, 1.0, st);
    }
  }

  // Each value appears once per column

  for (int j = 0; j < n; j++) {
    for (int v = 0; v < n; v++) {
      expr = new GRBLinExpr();
      for (int i = 0; i < n; i++)
        expr.addTerm(1.0, vars[i][j][v]);
      String st = "C_" + String.valueOf(j) + "_" + String.valueOf(v);
      model.addConstr(expr, GRB.EQUAL, 1.0, st);
    }
  }
```

```
// Each value appears once per sub-grid

for (int v = 0; v < n; v++) {
  for (int i0 = 0; i0 < s; i0++) {
    for (int j0 = 0; j0 < s; j0++) {
      expr = new GRBLinExpr();
      for (int i1 = 0; i1 < s; i1++) {
        for (int j1 = 0; j1 < s; j1++) {
          expr.addTerm(1.0, vars[i0*s+i1][j0*s+j1][v]);
        }
      }
      String st = "Sub_" + String.valueOf(v) + "_" + String.valueOf(i0)
                         + "_" + String.valueOf(j0);
      model.addConstr(expr, GRB.EQUAL, 1.0, st);
    }
  }
}

// Fix variables associated with pre-specified cells

File file = new File(args[0]);
FileInputStream fis = new FileInputStream(file);
byte[] input = new byte[n];

for (int i = 0; i < n; i++) {
  fis.read(input);
  for (int j = 0; j < n; j++) {
    int val = (int) input[j] - 48 - 1; // 0-based

    if (val >= 0)
      vars[i][j][val].set(GRB.DoubleAttr.LB, 1.0);
  }
  // read the endline byte
  fis.read();
}

// Optimize model

model.optimize();

// Write model to file
model.write("sudoku.lp");

double[][][] x = model.get(GRB.DoubleAttr.X, vars);
```

```
      System.out.println();
      for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
          for (int v = 0; v < n; v++) {
            if (x[i][j][v] > 0.5) {
              System.out.print(v+1);
            }
          }
        }
        System.out.println();
      }

      // Dispose of model and environment
      model.dispose();
      env.dispose();

    } catch (GRBException e) {
      System.out.println("Error code: " + e.getErrorCode() + ". " +
          e.getMessage());
    } catch (IOException e) {
      System.out.println("IO Error");
    }
  }
}
```

Tsp.java

```
/* Copyright 2017, Gurobi Optimization, Inc. */

// Solve a traveling salesman problem on a randomly generated set of
// points using lazy constraints.   The base MIP model only includes
// 'degree-2' constraints, requiring each node to have exactly
// two incident edges.  Solutions to this model may contain subtours -
// tours that don't visit every node.  The lazy constraint callback
// adds new constraints to cut them off.

import gurobi.*;

public class Tsp extends GRBCallback {
  private GRBVar[][] vars;

  public Tsp(GRBVar[][] xvars) {
    vars = xvars;
  }

  // Subtour elimination callback.  Whenever a feasible solution is found,
  // find the subtour that contains node 0, and add a subtour elimination
  // constraint if the tour doesn't visit every node.

  protected void callback() {
    try {
      if (where == GRB.CB_MIPSOL) {
        // Found an integer feasible solution - does it visit every node?
        int n = vars.length;
        int[] tour = findsubtour(getSolution(vars));

        if (tour.length < n) {
          // Add subtour elimination constraint
          GRBLinExpr expr = new GRBLinExpr();
          for (int i = 0; i < tour.length; i++)
            for (int j = i+1; j < tour.length; j++)
              expr.addTerm(1.0, vars[tour[i]][tour[j]]);
          addLazy(expr, GRB.LESS_EQUAL, tour.length-1);
        }
      }
    } catch (GRBException e) {
      System.out.println("Error code: " + e.getErrorCode() + ". " +
          e.getMessage());
      e.printStackTrace();
    }
  }
```

```
// Given an integer-feasible solution 'sol', return the smallest
// sub-tour (as a list of node indices).

protected static int[] findsubtour(double[][] sol)
{
  int n = sol.length;
  boolean[] seen = new boolean[n];
  int[] tour = new int[n];
  int bestind, bestlen;
  int i, node, len, start;

  for (i = 0; i < n; i++)
    seen[i] = false;

  start = 0;
  bestlen = n+1;
  bestind = -1;
  node = 0;
  while (start < n) {
    for (node = 0; node < n; node++)
      if (!seen[node])
        break;
    if (node == n)
      break;
    for (len = 0; len < n; len++) {
      tour[start+len] = node;
      seen[node] = true;
      for (i = 0; i < n; i++) {
        if (sol[node][i] > 0.5 && !seen[i]) {
          node = i;
          break;
        }
      }
      if (i == n) {
        len++;
        if (len < bestlen) {
          bestlen = len;
          bestind = start;
        }
        start += len;
        break;
      }
    }
  }
```

```
  int result[] = new int[bestlen];
  for (i = 0; i < bestlen; i++)
    result[i] = tour[bestind+i];
  return result;
}

// Euclidean distance between points 'i' and 'j'

protected static double distance(double[] x,
                                 double[] y,
                                 int      i,
                                 int      j) {
  double dx = x[i]-x[j];
  double dy = y[i]-y[j];
  return Math.sqrt(dx*dx+dy*dy);
}

public static void main(String[] args) {

  if (args.length < 1) {
    System.out.println("Usage: java Tsp ncities");
    System.exit(1);
  }

  int n = Integer.parseInt(args[0]);

  try {
    GRBEnv   env   = new GRBEnv();
    GRBModel model = new GRBModel(env);

    // Must set LazyConstraints parameter when using lazy constraints

    model.set(GRB.IntParam.LazyConstraints, 1);

    double[] x = new double[n];
    double[] y = new double[n];

    for (int i = 0; i < n; i++) {
      x[i] = Math.random();
      y[i] = Math.random();
    }

    // Create variables
```

```
  GRBVar[][] vars = new GRBVar[n][n];

  for (int i = 0; i < n; i++)
    for (int j = 0; j <= i; j++) {
      vars[i][j] = model.addVar(0.0, 1.0, distance(x, y, i, j),
                                GRB.BINARY,
                               "x"+String.valueOf(i)+"_"+String.valueOf(j));
      vars[j][i] = vars[i][j];
    }

  // Degree-2 constraints

  for (int i = 0; i < n; i++) {
    GRBLinExpr expr = new GRBLinExpr();
    for (int j = 0; j < n; j++)
      expr.addTerm(1.0, vars[i][j]);
    model.addConstr(expr, GRB.EQUAL, 2.0, "deg2_"+String.valueOf(i));
  }

  // Forbid edge from node back to itself

  for (int i = 0; i < n; i++)
    vars[i][i].set(GRB.DoubleAttr.UB, 0.0);

  model.setCallback(new Tsp(vars));
  model.optimize();

  if (model.get(GRB.IntAttr.SolCount) > 0) {
    int[] tour = findsubtour(model.get(GRB.DoubleAttr.X, vars));
    assert tour.length == n;

    System.out.print("Tour: ");
    for (int i = 0; i < tour.length; i++)
      System.out.print(String.valueOf(tour[i]) + " ");
    System.out.println();
  }

  // Dispose of model and environment
  model.dispose();
  env.dispose();

} catch (GRBException e) {
  System.out.println("Error code: " + e.getErrorCode() + ". " +
      e.getMessage());
  e.printStackTrace();
```

```
    }
  }
}
```

Tune.java

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* This example reads a model from a file and tunes it.
   It then writes the best parameter settings to a file
   and solves the model using these parameters. */

import gurobi.*;

public class Tune {
  public static void main(String[] args) {

    if (args.length < 1) {
      System.out.println("Usage: java Tune filename");
      System.exit(1);
    }

    try {
      GRBEnv env = new GRBEnv();

      // Read model from file
      GRBModel model = new GRBModel(env, args[0]);

      // Set the TuneResults parameter to 1
      model.set(GRB.IntParam.TuneResults, 1);

      // Tune the model
      model.tune();

      // Get the number of tuning results
      int resultcount = model.get(GRB.IntAttr.TuneResultCount);

      if (resultcount > 0) {

        // Load the tuned parameters into the model's environment
        model.getTuneResult(0);

        // Write the tuned parameters to a file
        model.write("tune.prm");

        // Solve the model using the tuned parameters
        model.optimize();
      }

      // Dispose of model and environment
```

```
      model.dispose();
      env.dispose();

    } catch (GRBException e) {
      System.out.println("Error code: " + e.getErrorCode() + ". "
          + e.getMessage());
    }
  }
}
```

Workforce1.java

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* Assign workers to shifts; each worker may or may not be available on a
   particular day. If the problem cannot be solved, use IIS to find a set of
   conflicting constraints. Note that there may be additional conflicts
   besides what is reported via IIS. */

import gurobi.*;

public class Workforce1 {

  public static void main(String[] args) {
    try {

      // Sample data
      // Sets of days and workers
      String Shifts[] =
          new String[] { "Mon1", "Tue2", "Wed3", "Thu4", "Fri5", "Sat6",
              "Sun7", "Mon8", "Tue9", "Wed10", "Thu11", "Fri12", "Sat13",
              "Sun14" };
      String Workers[] =
          new String[] { "Amy", "Bob", "Cathy", "Dan", "Ed", "Fred", "Gu" };

      int nShifts = Shifts.length;
      int nWorkers = Workers.length;

      // Number of workers required for each shift
      double shiftRequirements[] =
          new double[] { 3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5 };

      // Amount each worker is paid to work one shift
      double pay[] = new double[] { 10, 12, 10, 8, 8, 9, 11 };

      // Worker availability: 0 if the worker is unavailable for a shift
      double availability[][] =
          new double[][] { { 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1 },
              { 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0 },
              { 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
              { 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1 },
              { 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1 },
              { 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1 },
              { 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 } };

      // Model
```

```
GRBEnv env = new GRBEnv();
GRBModel model = new GRBModel(env);
model.set(GRB.StringAttr.ModelName, "assignment");

// Assignment variables: x[w][s] == 1 if worker w is assigned
// to shift s. Since an assignment model always produces integer
// solutions, we use continuous variables and solve as an LP.
GRBVar[][] x = new GRBVar[nWorkers][nShifts];
for (int w = 0; w < nWorkers; ++w) {
  for (int s = 0; s < nShifts; ++s) {
    x[w][s] =
        model.addVar(0, availability[w][s], pay[w], GRB.CONTINUOUS,
                     Workers[w] + "." + Shifts[s]);
  }
}

// The objective is to minimize the total pay costs
model.set(GRB.IntAttr.ModelSense, GRB.MINIMIZE);

// Constraint: assign exactly shiftRequirements[s] workers
// to each shift s
for (int s = 0; s < nShifts; ++s) {
  GRBLinExpr lhs = new GRBLinExpr();
  for (int w = 0; w < nWorkers; ++w) {
    lhs.addTerm(1.0, x[w][s]);
  }
  model.addConstr(lhs, GRB.EQUAL, shiftRequirements[s], Shifts[s]);
}

// Optimize
model.optimize();
int status = model.get(GRB.IntAttr.Status);
if (status == GRB.Status.UNBOUNDED) {
  System.out.println("The model cannot be solved "
      + "because it is unbounded");
  return;
}
if (status == GRB.Status.OPTIMAL) {
  System.out.println("The optimal objective is " +
      model.get(GRB.DoubleAttr.ObjVal));
  return;
}
if (status != GRB.Status.INF_OR_UNBD &&
    status != GRB.Status.INFEASIBLE    ){
  System.out.println("Optimization was stopped with status " + status);
```

```
        return;
      }

      // Compute IIS
      System.out.println("The model is infeasible; computing IIS");
      model.computeIIS();
      System.out.println("\nThe following constraint(s) "
          + "cannot be satisfied:");
      for (GRBConstr c : model.getConstrs()) {
        if (c.get(GRB.IntAttr.IISConstr) == 1) {
          System.out.println(c.get(GRB.StringAttr.ConstrName));
        }
      }

      // Dispose of model and environment
      model.dispose();
      env.dispose();

    } catch (GRBException e) {
      System.out.println("Error code: " + e.getErrorCode() + ". " +
          e.getMessage());
    }
  }
}
```

Workforce2.java

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* Assign workers to shifts; each worker may or may not be available on a
   particular day. If the problem cannot be solved, use IIS iteratively to
   find all conflicting constraints. */

import gurobi.*;
import java.util.*;

public class Workforce2 {

  public static void main(String[] args) {
    try {

      // Sample data
      // Sets of days and workers
      String Shifts[] =
          new String[] { "Mon1", "Tue2", "Wed3", "Thu4", "Fri5", "Sat6",
              "Sun7", "Mon8", "Tue9", "Wed10", "Thu11", "Fri12", "Sat13",
              "Sun14" };
      String Workers[] =
          new String[] { "Amy", "Bob", "Cathy", "Dan", "Ed", "Fred", "Gu" };

      int nShifts = Shifts.length;
      int nWorkers = Workers.length;

      // Number of workers required for each shift
      double shiftRequirements[] =
          new double[] { 3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5 };

      // Amount each worker is paid to work one shift
      double pay[] = new double[] { 10, 12, 10, 8, 8, 9, 11 };

      // Worker availability: 0 if the worker is unavailable for a shift
      double availability[][] =
          new double[][] { { 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1 },
              { 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0 },
              { 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
              { 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1 },
              { 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1 },
              { 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1 },
              { 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 } };

      // Model
```

```
GRBEnv env = new GRBEnv();
GRBModel model = new GRBModel(env);
model.set(GRB.StringAttr.ModelName, "assignment");

// Assignment variables: x[w][s] == 1 if worker w is assigned
// to shift s. Since an assignment model always produces integer
// solutions, we use continuous variables and solve as an LP.
GRBVar[][] x = new GRBVar[nWorkers][nShifts];
for (int w = 0; w < nWorkers; ++w) {
  for (int s = 0; s < nShifts; ++s) {
    x[w][s] =
        model.addVar(0, availability[w][s], pay[w], GRB.CONTINUOUS,
                     Workers[w] + "." + Shifts[s]);
  }
}

// The objective is to minimize the total pay costs
model.set(GRB.IntAttr.ModelSense, GRB.MINIMIZE);

// Constraint: assign exactly shiftRequirements[s] workers
// to each shift s
for (int s = 0; s < nShifts; ++s) {
  GRBLinExpr lhs = new GRBLinExpr();
  for (int w = 0; w < nWorkers; ++w) {
    lhs.addTerm(1.0, x[w][s]);
  }
  model.addConstr(lhs, GRB.EQUAL, shiftRequirements[s], Shifts[s]);
}

// Optimize
model.optimize();
int status = model.get(GRB.IntAttr.Status);
if (status == GRB.Status.UNBOUNDED) {
  System.out.println("The model cannot be solved "
      + "because it is unbounded");
  return;
}
if (status == GRB.Status.OPTIMAL) {
  System.out.println("The optimal objective is " +
      model.get(GRB.DoubleAttr.ObjVal));
  return;
}
if (status != GRB.Status.INF_OR_UNBD &&
    status != GRB.Status.INFEASIBLE    ) {
  System.out.println("Optimization was stopped with status " + status);
```

```
  return;
}

// Do IIS
System.out.println("The model is infeasible; computing IIS");
LinkedList<String> removed = new LinkedList<String>();

// Loop until we reduce to a model that can be solved
while (true) {
  model.computeIIS();
  System.out.println("\nThe following constraint cannot be satisfied:");
  for (GRBConstr c : model.getConstrs()) {
    if (c.get(GRB.IntAttr.IISConstr) == 1) {
      System.out.println(c.get(GRB.StringAttr.ConstrName));
      // Remove a single constraint from the model
      removed.add(c.get(GRB.StringAttr.ConstrName));
      model.remove(c);
      break;
    }
  }

  System.out.println();
  model.optimize();
  status = model.get(GRB.IntAttr.Status);

  if (status == GRB.Status.UNBOUNDED) {
    System.out.println("The model cannot be solved "
        + "because it is unbounded");
    return;
  }
  if (status == GRB.Status.OPTIMAL) {
    break;
  }
  if (status != GRB.Status.INF_OR_UNBD &&
      status != GRB.Status.INFEASIBLE    ) {
    System.out.println("Optimization was stopped with status " +
        status);
    return;
  }
}

System.out.println("\nThe following constraints were removed "
    + "to get a feasible LP:");
for (String s : removed) {
  System.out.print(s + " ");
```

```
      }
      System.out.println();

      // Dispose of model and environment
      model.dispose();
      env.dispose();

    } catch (GRBException e) {
      System.out.println("Error code: " + e.getErrorCode() + ". " +
          e.getMessage());
    }
  }
}
```

Workforce3.java

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* Assign workers to shifts; each worker may or may not be available on a
   particular day. If the problem cannot be solved, relax the model
   to determine which constraints cannot be satisfied, and how much
   they need to be relaxed. */

import gurobi.*;

public class Workforce3 {

  public static void main(String[] args) {
    try {

      // Sample data
      // Sets of days and workers
      String Shifts[] =
          new String[] { "Mon1", "Tue2", "Wed3", "Thu4", "Fri5", "Sat6",
              "Sun7", "Mon8", "Tue9", "Wed10", "Thu11", "Fri12", "Sat13",
              "Sun14" };
      String Workers[] =
          new String[] { "Amy", "Bob", "Cathy", "Dan", "Ed", "Fred", "Gu" };

      int nShifts = Shifts.length;
      int nWorkers = Workers.length;

      // Number of workers required for each shift
      double shiftRequirements[] =
          new double[] { 3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5 };

      // Amount each worker is paid to work one shift
      double pay[] = new double[] { 10, 12, 10, 8, 8, 9, 11 };

      // Worker availability: 0 if the worker is unavailable for a shift
      double availability[][] =
          new double[][] { { 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1 },
              { 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0 },
              { 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
              { 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1 },
              { 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1 },
              { 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1 },
              { 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 } };

      // Model
```

```
GRBEnv env = new GRBEnv();
GRBModel model = new GRBModel(env);
model.set(GRB.StringAttr.ModelName, "assignment");

// Assignment variables: x[w][s] == 1 if worker w is assigned
// to shift s. Since an assignment model always produces integer
// solutions, we use continuous variables and solve as an LP.
GRBVar[][] x = new GRBVar[nWorkers][nShifts];
for (int w = 0; w < nWorkers; ++w) {
  for (int s = 0; s < nShifts; ++s) {
    x[w][s] =
        model.addVar(0, availability[w][s], pay[w], GRB.CONTINUOUS,
                     Workers[w] + "." + Shifts[s]);
  }
}

// The objective is to minimize the total pay costs
model.set(GRB.IntAttr.ModelSense, GRB.MINIMIZE);

// Constraint: assign exactly shiftRequirements[s] workers
// to each shift s
for (int s = 0; s < nShifts; ++s) {
  GRBLinExpr lhs = new GRBLinExpr();
  for (int w = 0; w < nWorkers; ++w) {
    lhs.addTerm(1.0, x[w][s]);
  }
  model.addConstr(lhs, GRB.EQUAL, shiftRequirements[s], Shifts[s]);
}

// Optimize
model.optimize();
int status = model.get(GRB.IntAttr.Status);
if (status == GRB.UNBOUNDED) {
  System.out.println("The model cannot be solved "
      + "because it is unbounded");
  return;
}
if (status == GRB.OPTIMAL) {
  System.out.println("The optimal objective is " +
      model.get(GRB.DoubleAttr.ObjVal));
  return;
}
if (status != GRB.INF_OR_UNBD &&
    status != GRB.INFEASIBLE    ) {
  System.out.println("Optimization was stopped with status " + status);
```

```
        return;
      }

      // Relax the constraints to make the model feasible
      System.out.println("The model is infeasible; relaxing the constraints");
      int orignumvars = model.get(GRB.IntAttr.NumVars);
      model.feasRelax(0, false, false, true);
      model.optimize();
      status = model.get(GRB.IntAttr.Status);
      if (status == GRB.INF_OR_UNBD ||
          status == GRB.INFEASIBLE  ||
          status == GRB.UNBOUNDED     ) {
        System.out.println("The relaxed model cannot be solved "
            + "because it is infeasible or unbounded");
        return;
      }
      if (status != GRB.OPTIMAL) {
        System.out.println("Optimization was stopped with status " + status);
        return;
      }

      System.out.println("\nSlack values:");
      GRBVar[] vars = model.getVars();
      for (int i = orignumvars; i < model.get(GRB.IntAttr.NumVars); ++i) {
        GRBVar sv = vars[i];
        if (sv.get(GRB.DoubleAttr.X) > 1e-6) {
          System.out.println(sv.get(GRB.StringAttr.VarName) + " = " +
              sv.get(GRB.DoubleAttr.X));
        }
      }

      // Dispose of model and environment
      model.dispose();
      env.dispose();

    } catch (GRBException e) {
      System.out.println("Error code: " + e.getErrorCode() + ". " +
          e.getMessage());
    }
  }
}
```

Workforce4.java

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* Assign workers to shifts; each worker may or may not be available on a
   particular day. We use Pareto optimization to solve the model:
   first, we minimize the linear sum of the slacks. Then, we constrain
   the sum of the slacks, and we minimize a quadratic objective that
   tries to balance the workload among the workers. */

import gurobi.*;

public class Workforce4 {

  public static void main(String[] args) {
    try {

      // Sample data
      // Sets of days and workers
      String Shifts[] =
          new String[] { "Mon1", "Tue2", "Wed3", "Thu4", "Fri5", "Sat6",
              "Sun7", "Mon8", "Tue9", "Wed10", "Thu11", "Fri12", "Sat13",
              "Sun14" };
      String Workers[] =
          new String[] { "Amy", "Bob", "Cathy", "Dan", "Ed", "Fred", "Gu" };

      int nShifts = Shifts.length;
      int nWorkers = Workers.length;

      // Number of workers required for each shift
      double shiftRequirements[] =
          new double[] { 3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5 };

      // Worker availability: 0 if the worker is unavailable for a shift
      double availability[][] =
          new double[][] { { 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1 },
              { 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0 },
              { 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
              { 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1 },
              { 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1 },
              { 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1 },
              { 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 } };

      // Model
      GRBEnv env = new GRBEnv();
      GRBModel model = new GRBModel(env);
```

```
model.set(GRB.StringAttr.ModelName, "assignment");

// Assignment variables: x[w][s] == 1 if worker w is assigned
// to shift s. This is no longer a pure assignment model, so we must
// use binary variables.
GRBVar[][] x = new GRBVar[nWorkers][nShifts];
for (int w = 0; w < nWorkers; ++w) {
  for (int s = 0; s < nShifts; ++s) {
    x[w][s] =
        model.addVar(0, availability[w][s], 0, GRB.BINARY,
                     Workers[w] + "." + Shifts[s]);
  }
}

// Slack variables for each shift constraint so that the shifts can
// be satisfied
GRBVar[] slacks = new GRBVar[nShifts];
for (int s = 0; s < nShifts; ++s) {
  slacks[s] =
      model.addVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS,
                   Shifts[s] + "Slack");
}

// Variable to represent the total slack
GRBVar totSlack = model.addVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS,
                               "totSlack");

// Variables to count the total shifts worked by each worker
GRBVar[] totShifts = new GRBVar[nWorkers];
for (int w = 0; w < nWorkers; ++w) {
  totShifts[w] = model.addVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS,
                              Workers[w] + "TotShifts");
}

GRBLinExpr lhs;

// Constraint: assign exactly shiftRequirements[s] workers
// to each shift s, plus the slack
for (int s = 0; s < nShifts; ++s) {
  lhs = new GRBLinExpr();
  lhs.addTerm(1.0, slacks[s]);
  for (int w = 0; w < nWorkers; ++w) {
    lhs.addTerm(1.0, x[w][s]);
  }
  model.addConstr(lhs, GRB.EQUAL, shiftRequirements[s], Shifts[s]);
```

```
}

// Constraint: set totSlack equal to the total slack
lhs = new GRBLinExpr();
lhs.addTerm(-1.0, totSlack);
for (int s = 0; s < nShifts; ++s) {
  lhs.addTerm(1.0, slacks[s]);
}
model.addConstr(lhs, GRB.EQUAL, 0, "totSlack");

// Constraint: compute the total number of shifts for each worker
for (int w = 0; w < nWorkers; ++w) {
  lhs = new GRBLinExpr();
  lhs.addTerm(-1.0, totShifts[w]);
  for (int s = 0; s < nShifts; ++s) {
    lhs.addTerm(1.0, x[w][s]);
  }
  model.addConstr(lhs, GRB.EQUAL, 0, "totShifts" + Workers[w]);
}

// Objective: minimize the total slack
GRBLinExpr obj = new GRBLinExpr();
obj.addTerm(1.0, totSlack);
model.setObjective(obj);

// Optimize
int status =
  solveAndPrint(model, totSlack, nWorkers, Workers, totShifts);
if (status != GRB.Status.OPTIMAL ) {
  return;
}

// Constrain the slack by setting its upper and lower bounds
totSlack.set(GRB.DoubleAttr.UB, totSlack.get(GRB.DoubleAttr.X));
totSlack.set(GRB.DoubleAttr.LB, totSlack.get(GRB.DoubleAttr.X));

// Variable to count the average number of shifts worked
GRBVar avgShifts =
  model.addVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS, "avgShifts");

// Variables to count the difference from average for each worker;
// note that these variables can take negative values.
GRBVar[] diffShifts = new GRBVar[nWorkers];
for (int w = 0; w < nWorkers; ++w) {
  diffShifts[w] = model.addVar(-GRB.INFINITY, GRB.INFINITY, 0,
```

```
                                    GRB.CONTINUOUS, Workers[w] + "Diff");
    }

    // Constraint: compute the average number of shifts worked
    lhs = new GRBLinExpr();
    lhs.addTerm(-nWorkers, avgShifts);
    for (int w = 0; w < nWorkers; ++w) {
      lhs.addTerm(1.0, totShifts[w]);
    }
    model.addConstr(lhs, GRB.EQUAL, 0, "avgShifts");

    // Constraint: compute the difference from the average number of shifts
    for (int w = 0; w < nWorkers; ++w) {
      lhs = new GRBLinExpr();
      lhs.addTerm(-1, diffShifts[w]);
      lhs.addTerm(-1, avgShifts);
      lhs.addTerm( 1, totShifts[w]);
      model.addConstr(lhs, GRB.EQUAL, 0, Workers[w] + "Diff");
    }

    // Objective: minimize the sum of the square of the difference from the
    // average number of shifts worked
    GRBQuadExpr qobj = new GRBQuadExpr();
    for (int w = 0; w < nWorkers; ++w) {
      qobj.addTerm(1.0, diffShifts[w], diffShifts[w]);
    }
    model.setObjective(qobj);

    // Optimize
    status =
      solveAndPrint(model, totSlack, nWorkers, Workers, totShifts);
    if (status != GRB.Status.OPTIMAL ) {
      return;
    }

    // Dispose of model and environment
    model.dispose();
    env.dispose();

  } catch (GRBException e) {
    System.out.println("Error code: " + e.getErrorCode() + ". " +
        e.getMessage());
  }
}
```

```
  private static int solveAndPrint(GRBModel model, GRBVar totSlack,
                                   int nWorkers, String[] Workers,
                                   GRBVar[] totShifts) throws GRBException {

    model.optimize();
    int status = model.get(GRB.IntAttr.Status);
    if (status == GRB.Status.INF_OR_UNBD ||
        status == GRB.Status.INFEASIBLE  ||
        status == GRB.Status.UNBOUNDED     ) {
      System.out.println("The model cannot be solved "
          + "because it is infeasible or unbounded");
      return status;
    }
    if (status != GRB.Status.OPTIMAL ) {
      System.out.println("Optimization was stopped with status " + status);
      return status;
    }

    // Print total slack and the number of shifts worked for each worker
    System.out.println("\nTotal slack required: " +
                       totSlack.get(GRB.DoubleAttr.X));
    for (int w = 0; w < nWorkers; ++w) {
      System.out.println(Workers[w] + " worked " +
                         totShifts[w].get(GRB.DoubleAttr.X) + " shifts");
    }
    System.out.println("\n");
    return status;
  }

}
```

Workforce5.java

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* Assign workers to shifts; each worker may or may not be available on a
   particular day. We use multi-objective optimization to solve the model.
   The highest-priority objective minimizes the sum of the slacks
   (i.e., the total number of uncovered shifts). The secondary objective
   minimizes the difference between the maximum and minimum number of
   shifts worked among all workers.  The second optimization is allowed
   to degrade the first objective by up to the smaller value of 10% and 2 */

import gurobi.*;

public class Workforce5 {

  public static void main(String[] args) {

    try {

      // Sample data
      // Sets of days and workers
      String Shifts[] =
          new String[]  { "Mon1", "Tue2", "Wed3", "Thu4", "Fri5", "Sat6",
              "Sun7", "Mon8", "Tue9", "Wed10", "Thu11", "Fri12", "Sat13",
              "Sun14" };
      String Workers[] =
          new String[] { "Amy", "Bob", "Cathy", "Dan", "Ed", "Fred", "Gu", "Tobi" };

      int nShifts = Shifts.length;
      int nWorkers = Workers.length;

      // Number of workers required for each shift
      double shiftRequirements[] =
          new double[] { 3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5 };

      // Worker availability: 0 if the worker is unavailable for a shift
      double availability[][] =
          new double[][] { { 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1 },
              { 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0 },
              { 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
              { 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1 },
              { 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1 },
              { 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1 },
              { 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1 },
              { 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 } };
```

```
// Create environment
GRBEnv env = new GRBEnv();

// Create initial model
GRBModel model = new GRBModel(env);
model.set(GRB.StringAttr.ModelName, "Workforce5");

// Initialize assignment decision variables:
// x[w][s] == 1 if worker w is assigned to shift s.
// This is no longer a pure assignment model, so we must
// use binary variables.
GRBVar[][] x = new GRBVar[nWorkers][nShifts];
for (int w = 0; w < nWorkers; ++w) {
  for (int s = 0; s < nShifts; ++s) {
    x[w][s] =
        model.addVar(0, availability[w][s], 0, GRB.BINARY,
                     Workers[w] + "." + Shifts[s]);
  }
}

// Slack variables for each shift constraint so that the shifts can
// be satisfied
GRBVar[] slacks = new GRBVar[nShifts];
for (int s = 0; s < nShifts; ++s) {
  slacks[s] =
      model.addVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS,
                   Shifts[s] + "Slack");
}

// Variable to represent the total slack
GRBVar totSlack = model.addVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS,
                               "totSlack");

// Variables to count the total shifts worked by each worker
GRBVar[] totShifts = new GRBVar[nWorkers];
for (int w = 0; w < nWorkers; ++w) {
  totShifts[w] = model.addVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS,
                              Workers[w] + "TotShifts");
}

GRBLinExpr lhs;

// Constraint: assign exactly shiftRequirements[s] workers
// to each shift s, plus the slack
```

```
for (int s = 0; s < nShifts; ++s) {
  lhs = new GRBLinExpr();
  lhs.addTerm(1.0, slacks[s]);
  for (int w = 0; w < nWorkers; ++w) {
    lhs.addTerm(1.0, x[w][s]);
  }
  model.addConstr(lhs, GRB.EQUAL, shiftRequirements[s], Shifts[s]);
}

// Constraint: set totSlack equal to the total slack
lhs = new GRBLinExpr();
lhs.addTerm(-1.0, totSlack);
for (int s = 0; s < nShifts; ++s) {
  lhs.addTerm(1.0, slacks[s]);
}
model.addConstr(lhs, GRB.EQUAL, 0, "totSlack");

// Constraint: compute the total number of shifts for each worker
for (int w = 0; w < nWorkers; ++w) {
  lhs = new GRBLinExpr();
  lhs.addTerm(-1.0, totShifts[w]);
  for (int s = 0; s < nShifts; ++s) {
    lhs.addTerm(1.0, x[w][s]);
  }
  model.addConstr(lhs, GRB.EQUAL, 0, "totShifts" + Workers[w]);
}

// Constraint: set minShift/maxShift variable to less <=/>= to the
// number of shifts among all workers
GRBVar minShift = model.addVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS,
                               "minShift");
GRBVar maxShift = model.addVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS,
                               "maxShift");
model.addGenConstrMin(minShift, totShifts, GRB.INFINITY, "minShift");
model.addGenConstrMax(maxShift, totShifts, -GRB.INFINITY, "maxShift");

// Set global sense for ALL objectives
model.set(GRB.IntAttr.ModelSense, GRB.MINIMIZE);

// Set primary objective
GRBLinExpr obj0 = new GRBLinExpr();
obj0.addTerm(1.0, totSlack);
model.setObjectiveN(obj0, 0, 2, 1.0, 2.0, 0.1, "TotalSlack");

// Set secondary objective
```

```
    GRBLinExpr obj1 = new GRBLinExpr();
    obj1.addTerm(1.0, maxShift);
    obj1.addTerm(-1.0, minShift);
    model.setObjectiveN(obj1, 1, 1, 1.0, 0.0, 0.0, "Fairness");

    // Save problem
    model.write("Workforce5.lp");

    // Optimize
    int status = solveAndPrint(model, totSlack, nWorkers, Workers, totShifts);

    if (status != GRB.OPTIMAL)
      return;

    // Dispose of model and environment
    model.dispose();
    env.dispose();

  } catch (GRBException e) {
    System.out.println("Error code: " + e.getErrorCode() + ". " +
        e.getMessage());
  }
}

private static int solveAndPrint(GRBModel model, GRBVar totSlack,
                                 int nWorkers, String[] Workers,
                                 GRBVar[] totShifts) throws GRBException {

  model.optimize();
  int status = model.get(GRB.IntAttr.Status);
  if (status == GRB.Status.INF_OR_UNBD ||
      status == GRB.Status.INFEASIBLE  ||
      status == GRB.Status.UNBOUNDED     ) {
    System.out.println("The model cannot be solved "
        + "because it is infeasible or unbounded");
    return status;
  }
  if (status != GRB.Status.OPTIMAL ) {
    System.out.println("Optimization was stopped with status " + status);
    return status;
  }

  // Print total slack and the number of shifts worked for each worker
  System.out.println("\nTotal slack required: " +
                     totSlack.get(GRB.DoubleAttr.X));
```

```
    for (int w = 0; w < nWorkers; ++w) {
      System.out.println(Workers[w] + " worked " +
                         totShifts[w].get(GRB.DoubleAttr.X) + " shifts");
    }
    System.out.println("\n");
    return status;
  }

}
```

3.4 C# Examples

This section includes source code for all of the Gurobi C# examples. The same source code can be found in the `examples/c#` directory of the Gurobi distribution.

callback_cs.cs

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/*
   This example reads a model from a file, sets up a callback that
   monitors optimization progress and implements a custom
   termination strategy, and outputs progress information to the
   screen and to a log file.

   The termination strategy implemented in this callback stops the
   optimization of a MIP model once at least one of the following two
   conditions have been satisfied:
     1) The optimality gap is less than 10%
     2) At least 10000 nodes have been explored, and an integer feasible
        solution has been found.
   Note that termination is normally handled through Gurobi parameters
   (MIPGap, NodeLimit, etc.).  You should only use a callback for
   termination if the available parameters don't capture your desired
   termination criterion.
*/

using System;
using System.IO;
using Gurobi;

class callback_cs : GRBCallback
{
  private double       lastiter;
  private double       lastnode;
  private GRBVar[]     vars;
  private StreamWriter logfile;

  public callback_cs(GRBVar[] xvars, StreamWriter xlogfile)
  {
    lastiter = lastnode = -GRB.INFINITY;
    vars = xvars;
    logfile = xlogfile;
  }

  protected override void Callback()
```

```
{
  try {
    if (where == GRB.Callback.POLLING) {
      // Ignore polling callback
    } else if (where == GRB.Callback.PRESOLVE) {
      // Presolve callback
      int cdels = GetIntInfo(GRB.Callback.PRE_COLDEL);
      int rdels = GetIntInfo(GRB.Callback.PRE_ROWDEL);
      if (cdels != 0 || rdels != 0) {
        Console.WriteLine(cdels + " columns and " + rdels
            + " rows are removed");
      }
    } else if (where == GRB.Callback.SIMPLEX) {
      // Simplex callback
      double itcnt = GetDoubleInfo(GRB.Callback.SPX_ITRCNT);
      if (itcnt - lastiter >= 100) {
        lastiter = itcnt;
        double obj    = GetDoubleInfo(GRB.Callback.SPX_OBJVAL);
        int    ispert = GetIntInfo(GRB.Callback.SPX_ISPERT);
        double pinf   = GetDoubleInfo(GRB.Callback.SPX_PRIMINF);
        double dinf   = GetDoubleInfo(GRB.Callback.SPX_DUALINF);
        char ch;
        if (ispert == 0)      ch = ' ';
        else if (ispert == 1) ch = 'S';
        else                  ch = 'P';
        Console.WriteLine(itcnt + " " + obj + ch + " "
            + pinf + " " + dinf);
      }
    } else if (where == GRB.Callback.MIP) {
      // General MIP callback
      double nodecnt = GetDoubleInfo(GRB.Callback.MIP_NODCNT);
      double objbst  = GetDoubleInfo(GRB.Callback.MIP_OBJBST);
      double objbnd  = GetDoubleInfo(GRB.Callback.MIP_OBJBND);
      int    solcnt  = GetIntInfo(GRB.Callback.MIP_SOLCNT);
      if (nodecnt - lastnode >= 100) {
        lastnode = nodecnt;
        int actnodes = (int) GetDoubleInfo(GRB.Callback.MIP_NODLFT);
        int itcnt    = (int) GetDoubleInfo(GRB.Callback.MIP_ITRCNT);
        int cutcnt   = GetIntInfo(GRB.Callback.MIP_CUTCNT);
        Console.WriteLine(nodecnt + " " + actnodes + " "
            + itcnt + " " + objbst + " " + objbnd + " "
            + solcnt + " " + cutcnt);
      }
      if (Math.Abs(objbst - objbnd) < 0.1 * (1.0 + Math.Abs(objbst))) {
        Console.WriteLine("Stop early - 10% gap achieved");
```

```
      Abort();
    }
    if (nodecnt >= 10000 && solcnt > 0) {
      Console.WriteLine("Stop early - 10000 nodes explored");
      Abort();
    }
  } else if (where == GRB.Callback.MIPSOL) {
    // MIP solution callback
    int      nodecnt = (int) GetDoubleInfo(GRB.Callback.MIPSOL_NODCNT);
    double   obj     = GetDoubleInfo(GRB.Callback.MIPSOL_OBJ);
    int      solcnt  = GetIntInfo(GRB.Callback.MIPSOL_SOLCNT);
    double[] x       = GetSolution(vars);
    Console.WriteLine("**** New solution at node " + nodecnt
        + ", obj " + obj + ", sol " + solcnt
        + ", x[0] = " + x[0] + " ****");
  } else if (where == GRB.Callback.MIPNODE) {
    // MIP node callback
    Console.WriteLine("**** New node ****");
    if (GetIntInfo(GRB.Callback.MIPNODE_STATUS) == GRB.Status.OPTIMAL) {
      double[] x = GetNodeRel(vars);
      SetSolution(vars, x);
    }
  } else if (where == GRB.Callback.BARRIER) {
    // Barrier callback
    int    itcnt   = GetIntInfo(GRB.Callback.BARRIER_ITRCNT);
    double primobj = GetDoubleInfo(GRB.Callback.BARRIER_PRIMOBJ);
    double dualobj = GetDoubleInfo(GRB.Callback.BARRIER_DUALOBJ);
    double priminf = GetDoubleInfo(GRB.Callback.BARRIER_PRIMINF);
    double dualinf = GetDoubleInfo(GRB.Callback.BARRIER_DUALINF);
    double cmpl    = GetDoubleInfo(GRB.Callback.BARRIER_COMPL);
    Console.WriteLine(itcnt + " " + primobj + " " + dualobj + " "
        + priminf + " " + dualinf + " " + cmpl);
  } else if (where == GRB.Callback.MESSAGE) {
    // Message callback
    string msg = GetStringInfo(GRB.Callback.MSG_STRING);
    if (msg != null) logfile.Write(msg);
  }
} catch (GRBException e) {
  Console.WriteLine("Error code: " + e.ErrorCode);
  Console.WriteLine(e.Message);
  Console.WriteLine(e.StackTrace);
} catch (Exception e) {
  Console.WriteLine("Error during callback");
  Console.WriteLine(e.StackTrace);
}
```

```
}

static void Main(string[] args)
{
  if (args.Length < 1) {
    Console.Out.WriteLine("Usage: callback_cs filename");
    return;
  }

  StreamWriter logfile = null;

  try {
    // Create environment
    GRBEnv env = new GRBEnv();

    // Read model from file
    GRBModel model = new GRBModel(env, args[0]);

    // Turn off display and heuristics
    model.Parameters.OutputFlag = 0;
    model.Parameters.Heuristics = 0.0;

    // Open log file
    logfile = new StreamWriter("cb.log");

    // Create a callback object and associate it with the model
    GRBVar[]    vars = model.GetVars();
    callback_cs cb   = new callback_cs(vars, logfile);

    model.SetCallback(cb);

    // Solve model and capture solution information
    model.Optimize();

    Console.WriteLine("");
    Console.WriteLine("Optimization complete");
    if (model.SolCount == 0) {
      Console.WriteLine("No solution found, optimization status = "
          + model.Status);
    } else {
      Console.WriteLine("Solution found, objective = " + model.ObjVal);

      string[] vnames = model.Get(GRB.StringAttr.VarName, vars);
      double[] x      = model.Get(GRB.DoubleAttr.X, vars);
```

```
        for (int j = 0; j < vars.Length; j++) {
          if (x[j] != 0.0) Console.WriteLine(vnames[j] + " " + x[j]);
        }
      }

      // Dispose of model and environment
      model.Dispose();
      env.Dispose();

    } catch (GRBException e) {
      Console.WriteLine("Error code: " + e.ErrorCode);
      Console.WriteLine(e.Message);
      Console.WriteLine(e.StackTrace);
    } catch (Exception e) {
      Console.WriteLine("Error during optimization");
      Console.WriteLine(e.Message);
      Console.WriteLine(e.StackTrace);
    } finally {
      // Close log file
      if (logfile != null) logfile.Close();
    }
  }
}
```

dense_cs.cs

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* This example formulates and solves the following simple QP model:

     minimize    x + y + x^2 + x*y + y^2 + y*z + z^2
     subject to  x + 2 y + 3 z >= 4
                 x +   y       >= 1

   The example illustrates the use of dense matrices to store A and Q
   (and dense vectors for the other relevant data).  We don't recommend
   that you use dense matrices, but this example may be helpful if you
   already have your data in this format.
*/

using System;
using Gurobi;

class dense_cs {

  protected static bool
    dense_optimize(GRBEnv    env,
                   int       rows,
                   int       cols,
                   double[]  c,      // linear portion of objective function
                   double[,] Q,      // quadratic portion of objective function
                   double[,] A,      // constraint matrix
                   char[]    sense,  // constraint senses
                   double[]  rhs,    // RHS vector
                   double[]  lb,     // variable lower bounds
                   double[]  ub,     // variable upper bounds
                   char[]    vtype,  // variable types (continuous, binary, etc.)
                   double[]  solution) {

    bool success = false;

    try {
      GRBModel model = new GRBModel(env);

      // Add variables to the model

      GRBVar[] vars = model.AddVars(lb, ub, null, vtype, null);

      // Populate A matrix
```

```
    for (int i = 0; i < rows; i++) {
      GRBLinExpr expr = new GRBLinExpr();
      for (int j = 0; j < cols; j++)
        if (A[i,j] != 0)
          expr.AddTerm(A[i,j], vars[j]); // Note: '+=' would be much slower
      model.AddConstr(expr, sense[i], rhs[i], "");
    }

    // Populate objective

    GRBQuadExpr obj = new GRBQuadExpr();
    if (Q != null) {
      for (int i = 0; i < cols; i++)
        for (int j = 0; j < cols; j++)
          if (Q[i,j] != 0)
            obj.AddTerm(Q[i,j], vars[i], vars[j]); // Note: '+=' would be much slower
      for (int j = 0; j < cols; j++)
        if (c[j] != 0)
          obj.AddTerm(c[j], vars[j]); // Note: '+=' would be much slower
      model.SetObjective(obj);
    }

    // Solve model

    model.Optimize();

    // Extract solution

    if (model.Status == GRB.Status.OPTIMAL) {
      success = true;

      for (int j = 0; j < cols; j++)
        solution[j] = vars[j].X;
    }

    model.Dispose();

  } catch (GRBException e) {
    Console.WriteLine("Error code: " + e.ErrorCode + ". " + e.Message);
  }

  return success;
}

public static void Main(String[] args) {
```

```
    try {
      GRBEnv env = new GRBEnv();

      double[] c = new double[] {1, 1, 0};
      double[,] Q = new double[,] {{1, 1, 0}, {0, 1, 1}, {0, 0, 1}};
      double[,] A = new double[,] {{1, 2, 3}, {1, 1, 0}};
      char[] sense = new char[] {'>', '>'};
      double[] rhs = new double[] {4, 1};
      double[] lb = new double[] {0, 0, 0};
      bool success;
      double[] sol = new double[3];

      success = dense_optimize(env, 2, 3, c, Q, A, sense, rhs,
                               lb, null, null, sol);

      if (success) {
        Console.WriteLine("x: " + sol[0] + ", y: " + sol[1] + ", z: " + sol[2]);
      }

      // Dispose of environment
      env.Dispose();

    } catch (GRBException e) {
      Console.WriteLine("Error code: " + e.ErrorCode + ". " + e.Message);
    }
  }
}
```

diet_cs.cs

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* Solve the classic diet model, showing how to add constraints
   to an existing model. */

using System;
using Gurobi;

class diet_cs
{
  static void Main()
  {
    try {

      // Nutrition guidelines, based on
      // USDA Dietary Guidelines for Americans, 2005
      // http://www.health.gov/DietaryGuidelines/dga2005/
      string[] Categories =
          new string[] { "calories", "protein", "fat", "sodium" };
      int nCategories = Categories.Length;
      double[] minNutrition = new double[] { 1800, 91, 0, 0 };
      double[] maxNutrition = new double[] { 2200, GRB.INFINITY, 65, 1779 };

      // Set of foods
      string[] Foods =
          new string[] { "hamburger", "chicken", "hot dog", "fries",
              "macaroni", "pizza", "salad", "milk", "ice cream" };
      int nFoods = Foods.Length;
      double[] cost =
          new double[] { 2.49, 2.89, 1.50, 1.89, 2.09, 1.99, 2.49, 0.89,
              1.59 };

      // Nutrition values for the foods
      double[,] nutritionValues = new double[,] {
          { 410, 24, 26, 730 },   // hamburger
          { 420, 32, 10, 1190 },  // chicken
          { 560, 20, 32, 1800 },  // hot dog
          { 380, 4, 19, 270 },    // fries
          { 320, 12, 10, 930 },   // macaroni
          { 320, 15, 12, 820 },   // pizza
          { 320, 31, 12, 1230 },  // salad
          { 100, 8, 2.5, 125 },   // milk
          { 330, 8, 10, 180 }     // ice cream
          };
```

```
// Model
GRBEnv env = new GRBEnv();
GRBModel model = new GRBModel(env);

model.ModelName = "diet";

// Create decision variables for the nutrition information,
// which we limit via bounds
GRBVar[] nutrition = new GRBVar[nCategories];
for (int i = 0; i < nCategories; ++i) {
  nutrition[i] =
      model.AddVar(minNutrition[i], maxNutrition[i], 0, GRB.CONTINUOUS,
                   Categories[i]);
}

// Create decision variables for the foods to buy
GRBVar[] buy = new GRBVar[nFoods];
for (int j = 0; j < nFoods; ++j) {
  buy[j] =
      model.AddVar(0, GRB.INFINITY, cost[j], GRB.CONTINUOUS, Foods[j]);
}

// The objective is to minimize the costs
model.ModelSense = GRB.MINIMIZE;

// Nutrition constraints
for (int i = 0; i < nCategories; ++i) {
  GRBLinExpr ntot = 0.0;
  for (int j = 0; j < nFoods; ++j)
    ntot.AddTerm(nutritionValues[j,i], buy[j]);
  model.AddConstr(ntot == nutrition[i], Categories[i]);
}

// Solve
model.Optimize();
PrintSolution(model, buy, nutrition);

Console.WriteLine("\nAdding constraint: at most 6 servings of dairy");
model.AddConstr(buy[7] + buy[8] <= 6.0, "limit_dairy");

// Solve
model.Optimize();
PrintSolution(model, buy, nutrition);
```

```
      // Dispose of model and env
      model.Dispose();
      env.Dispose();

    } catch (GRBException e) {
      Console.WriteLine("Error code: " + e.ErrorCode + ". " +
          e.Message);
    }
  }

  private static void PrintSolution(GRBModel model, GRBVar[] buy,
                                    GRBVar[] nutrition) {
    if (model.Status == GRB.Status.OPTIMAL) {
      Console.WriteLine("\nCost: " + model.ObjVal);
      Console.WriteLine("\nBuy:");
      for (int j = 0; j < buy.Length; ++j) {
        if (buy[j].X > 0.0001) {
          Console.WriteLine(buy[j].VarName + " " + buy[j].X);
        }
      }
      Console.WriteLine("\nNutrition:");
      for (int i = 0; i < nutrition.Length; ++i) {
        Console.WriteLine(nutrition[i].VarName + " " + nutrition[i].X);
      }
    } else {
      Console.WriteLine("No solution");
    }
  }
}
```

facility_cs.cs

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* Facility location: a company currently ships its product from 5 plants
   to 4 warehouses. It is considering closing some plants to reduce
   costs. What plant(s) should the company close, in order to minimize
   transportation and fixed costs?

   Based on an example from Frontline Systems:
   http://www.solver.com/disfacility.htm
   Used with permission.
 */

using System;
using Gurobi;

class facility_cs
{
  static void Main()
  {
    try {

      // Warehouse demand in thousands of units
      double[] Demand = new double[] { 15, 18, 14, 20 };

      // Plant capacity in thousands of units
      double[] Capacity = new double[] { 20, 22, 17, 19, 18 };

      // Fixed costs for each plant
      double[] FixedCosts =
          new double[] { 12000, 15000, 17000, 13000, 16000 };

      // Transportation costs per thousand units
      double[,] TransCosts =
          new double[,] { { 4000, 2000, 3000, 2500, 4500 },
              { 2500, 2600, 3400, 3000, 4000 },
              { 1200, 1800, 2600, 4100, 3000 },
              { 2200, 2600, 3100, 3700, 3200 } };

      // Number of plants and warehouses
      int nPlants = Capacity.Length;
      int nWarehouses = Demand.Length;

      // Model
      GRBEnv env = new GRBEnv();
```

```
GRBModel model = new GRBModel(env);

model.ModelName = "facility";

// Plant open decision variables: open[p] == 1 if plant p is open.
GRBVar[] open = new GRBVar[nPlants];
for (int p = 0; p < nPlants; ++p) {
  open[p] = model.AddVar(0, 1, FixedCosts[p], GRB.BINARY, "Open" + p);
}

// Transportation decision variables: how much to transport from
// a plant p to a warehouse w
GRBVar[,] transport = new GRBVar[nWarehouses,nPlants];
for (int w = 0; w < nWarehouses; ++w) {
  for (int p = 0; p < nPlants; ++p) {
    transport[w,p] =
        model.AddVar(0, GRB.INFINITY, TransCosts[w,p], GRB.CONTINUOUS,
                     "Trans" + p + "." + w);
  }
}

// The objective is to minimize the total fixed and variable costs
model.ModelSense = GRB.MINIMIZE;

// Production constraints
// Note that the right-hand limit sets the production to zero if
// the plant is closed
for (int p = 0; p < nPlants; ++p) {
  GRBLinExpr ptot = 0.0;
  for (int w = 0; w < nWarehouses; ++w)
    ptot.AddTerm(1.0, transport[w,p]);
  model.AddConstr(ptot <= Capacity[p] * open[p], "Capacity" + p);
}

// Demand constraints
for (int w = 0; w < nWarehouses; ++w) {
  GRBLinExpr dtot = 0.0;
  for (int p = 0; p < nPlants; ++p)
    dtot.AddTerm(1.0, transport[w,p]);
  model.AddConstr(dtot == Demand[w], "Demand" + w);
}

// Guess at the starting point: close the plant with the highest
// fixed costs; open all others
```

```
// First, open all plants
for (int p = 0; p < nPlants; ++p) {
  open[p].Start = 1.0;
}

// Now close the plant with the highest fixed cost
Console.WriteLine("Initial guess:");
double maxFixed = -GRB.INFINITY;
for (int p = 0; p < nPlants; ++p) {
  if (FixedCosts[p] > maxFixed) {
    maxFixed = FixedCosts[p];
  }
}
for (int p = 0; p < nPlants; ++p) {
  if (FixedCosts[p] == maxFixed) {
    open[p].Start = 0.0;
    Console.WriteLine("Closing plant " + p + "\n");
    break;
  }
}

// Use barrier to solve root relaxation
model.Parameters.Method = GRB.METHOD_BARRIER;

// Solve
model.Optimize();

// Print solution
Console.WriteLine("\nTOTAL COSTS: " + model.ObjVal);
Console.WriteLine("SOLUTION:");
for (int p = 0; p < nPlants; ++p) {
  if (open[p].X > 0.99) {
    Console.WriteLine("Plant " + p + " open:");
    for (int w = 0; w < nWarehouses; ++w) {
      if (transport[w,p].X > 0.0001) {
        Console.WriteLine("  Transport " +
            transport[w,p].X + " units to warehouse " + w);
      }
    }
  } else {
    Console.WriteLine("Plant " + p + " closed!");
  }
}

// Dispose of model and env
```

```
      model.Dispose();
      env.Dispose();

    } catch (GRBException e) {
      Console.WriteLine("Error code: " + e.ErrorCode + ". " + e.Message);
    }
  }
}
```

feasopt_cs.cs

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* This example reads a MIP model from a file, adds artificial
   variables to each constraint, and then minimizes the sum of the
   artificial variables.  A solution with objective zero corresponds
   to a feasible solution to the input model.
   We can also use FeasRelax feature to do it. In this example, we
   use minrelax=1, i.e. optimizing the returned model finds a solution
   that minimizes the original objective, but only from among those
   solutions that minimize the sum of the artificial variables. */

using Gurobi;
using System;

class feasopt_cs
{
  static void Main(string[] args)
  {
    if (args.Length < 1) {
      Console.Out.WriteLine("Usage: feasopt_cs filename");
      return;
    }

    try {
      GRBEnv env = new GRBEnv();
      GRBModel feasmodel = new GRBModel(env, args[0]);

      // Create a copy to use FeasRelax feature later */
      GRBModel feasmodel1 = new GRBModel(feasmodel);

      // Clear objective
      feasmodel.SetObjective(new GRBLinExpr());

      // Add slack variables
      GRBConstr[] c = feasmodel.GetConstrs();
      for (int i = 0; i < c.Length; ++i) {
        char sense = c[i].Sense;
        if (sense != '>') {
          GRBConstr[] constrs = new GRBConstr[] { c[i] };
          double[] coeffs = new double[] { -1 };
          feasmodel.AddVar(0.0, GRB.INFINITY, 1.0, GRB.CONTINUOUS, constrs,
                           coeffs, "ArtN_" + c[i].ConstrName);
        }
        if (sense != '<') {
```

```
          GRBConstr[] constrs = new GRBConstr[] { c[i] };
          double[] coeffs = new double[] { 1 };
          feasmodel.AddVar(0.0, GRB.INFINITY, 1.0, GRB.CONTINUOUS, constrs,
                           coeffs, "ArtP_" +
                               c[i].ConstrName);
        }
      }

      // Optimize modified model
      feasmodel.Optimize();
      feasmodel.Write("feasopt.lp");

      // Use FeasRelax feature */
      feasmodel1.FeasRelax(GRB.FEASRELAX_LINEAR, true, false, true);
      feasmodel1.Write("feasopt1.lp");
      feasmodel1.Optimize();

      // Dispose of model and env
      feasmodel1.Dispose();
      feasmodel.Dispose();
      env.Dispose();

    } catch (GRBException e) {
      Console.WriteLine("Error code: " + e.ErrorCode + ". " + e.Message);
    }
  }
}
```

fixanddive_cs.cs

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* Implement a simple MIP heuristic.  Relax the model,
   sort variables based on fractionality, and fix the 25% of
   the fractional variables that are closest to integer variables.
   Repeat until either the relaxation is integer feasible or
   linearly infeasible. */

using System;
using System.Collections.Generic;
using Gurobi;

class fixanddive_cs
{
  // Comparison class used to sort variable list based on relaxation
  // fractionality

  class FractionalCompare : IComparer<GRBVar>
  {
    public int Compare(GRBVar v1, GRBVar v2)
    {
      try {
        double sol1 = Math.Abs(v1.X);
        double sol2 = Math.Abs(v2.X);
        double frac1 = Math.Abs(sol1 - Math.Floor(sol1 + 0.5));
        double frac2 = Math.Abs(sol2 - Math.Floor(sol2 + 0.5));
        if (frac1 < frac2) {
          return -1;
        } else if (frac1 > frac2) {
          return 1;
        } else {
          return 0;
        }
      } catch (GRBException e) {
        Console.WriteLine("Error code: " + e.ErrorCode + ". " +
            e.Message);
      }
      return 0;
    }
  }

  static void Main(string[] args)
  {
    if (args.Length < 1) {
```

```
  Console.Out.WriteLine("Usage: fixanddive_cs filename");
  return;
}

try {
  // Read model
  GRBEnv env = new GRBEnv();
  GRBModel model = new GRBModel(env, args[0]);

  // Collect integer variables and relax them
  List<GRBVar> intvars = new List<GRBVar>();
  foreach (GRBVar v in model.GetVars()) {
    if (v.VType != GRB.CONTINUOUS) {
      intvars.Add(v);
      v.VType = GRB.CONTINUOUS;
    }
  }

  model.Parameters.OutputFlag = 0;
  model.Optimize();

  // Perform multiple iterations. In each iteration, identify the first
  // quartile of integer variables that are closest to an integer value
  // in the relaxation, fix them to the nearest integer, and repeat.

  for (int iter = 0; iter < 1000; ++iter) {

    // create a list of fractional variables, sorted in order of
    // increasing distance from the relaxation solution to the nearest
    // integer value

    List<GRBVar> fractional = new List<GRBVar>();
    foreach (GRBVar v in intvars) {
      double sol = Math.Abs(v.X);
      if (Math.Abs(sol - Math.Floor(sol + 0.5)) > 1e-5) {
        fractional.Add(v);
      }
    }

    Console.WriteLine("Iteration " + iter + ", obj " +
        model.ObjVal + ", fractional " + fractional.Count);

    if (fractional.Count == 0) {
      Console.WriteLine("Found feasible solution - objective " +
          model.ObjVal);
```

```
          break;
        }

        // Fix the first quartile to the nearest integer value

        fractional.Sort(new FractionalCompare());
        int nfix = Math.Max(fractional.Count / 4, 1);
        for (int i = 0; i < nfix; ++i) {
          GRBVar v = fractional[i];
          double fixval = Math.Floor(v.X + 0.5);
          v.LB = fixval;
          v.UB = fixval;
          Console.WriteLine("  Fix " + v.VarName +
              " to " + fixval + " ( rel " + v.X + " )");
        }

        model.Optimize();

        // Check optimization result

        if (model.Status != GRB.Status.OPTIMAL) {
          Console.WriteLine("Relaxation is infeasible");
          break;
        }
      }

      // Dispose of model and env
      model.Dispose();
      env.Dispose();

    } catch (GRBException e) {
      Console.WriteLine("Error code: " + e.ErrorCode + ". " +
          e.Message);
    }
  }
}
```

genconstr_cs.cs

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* In this example we show the use of general constraints for modeling
   some common expressions. We use as an example a SAT-problem where we
   want to see if it is possible to satisfy at least four (or all) clauses
   of the logical for

   L = (x0 or ~x1 or x2)  and (x1 or ~x2 or x3)  and
       (x2 or ~x3 or x0)  and (x3 or ~x0 or x1)  and
       (~x0 or ~x1 or x2) and (~x1 or ~x2 or x3) and
       (~x2 or ~x3 or x0) and (~x3 or ~x0 or x1)

   We do this by introducing two variables for each literal (itself and its
   negated value), a variable for each clause, and then two
   variables for indicating if we can satisfy four, and another to identify
   the minimum of the clauses (so if it one, we can satisfy all clauses)
   and put these two variables in the objective.
   i.e. the Objective function will be

   maximize Obj0 + Obj1

    Obj0 = MIN(Clause1, ... , Clause8)
    Obj1 = 1 -> Clause1 + ... + Clause8 >= 4

   thus, the objective value will be two if and only if we can satisfy all
   clauses; one if and only if at least four clauses can be satisfied, and
   zero otherwise.
 */

using System;
using Gurobi;

class genconstr_cs {

  public const int n = 4;
  public const int NLITERALS = 4;  // same as n
  public const int NCLAUSES = 8;
  public const int NOBJ = 2;

  static void Main() {

    try {
      // Example data:
      //   e.g. {0, n+1, 2} means clause (x0 or ~x1 or x2)
```

```
int[,] Clauses = new int[,]
                  {{  0, n+1, 2}, {  1, n+2, 3},
                   {  2, n+3, 0}, {  3, n+0, 1},
                   {n+0, n+1, 2}, {n+1, n+2, 3},
                   {n+2, n+3, 0}, {n+3, n+0, 1}};

int i, status;

// Create environment
GRBEnv env = new GRBEnv("genconstr_cs.log");

// Create initial model
GRBModel model = new GRBModel(env);
model.ModelName = "genconstr_cs";

// Initialize decision variables and objective

GRBVar[] Lit     = new GRBVar[NLITERALS];
GRBVar[] NotLit  = new GRBVar[NLITERALS];
for (i = 0; i < NLITERALS; i++) {
  Lit[i]    = model.AddVar(0.0, 1.0, 0.0, GRB.BINARY, string.Format("X{0}", i));
  NotLit[i] = model.AddVar(0.0, 1.0, 0.0, GRB.BINARY, string.Format("notX{0}", i));
}

GRBVar[] Cla = new GRBVar[NCLAUSES];
for (i = 0; i < NCLAUSES; i++) {
  Cla[i] = model.AddVar(0.0, 1.0, 0.0, GRB.BINARY, string.Format("Clause{0}", i));
}

GRBVar[] Obj = new GRBVar[NOBJ];
for (i = 0; i < NOBJ; i++) {
  Obj[i] = model.AddVar(0.0, 1.0, 1.0, GRB.BINARY, string.Format("Obj{0}", i));
}

// Link Xi and notXi
GRBLinExpr lhs;
for (i = 0; i < NLITERALS; i++) {
  lhs = new GRBLinExpr();
  lhs.AddTerm(1.0, Lit[i]);
  lhs.AddTerm(1.0, NotLit[i]);
  model.AddConstr(lhs, GRB.EQUAL, 1.0, string.Format("CNSTR_X{0}", i));
}

// Link clauses and literals
for (i = 0; i < NCLAUSES; i++) {
```

```
  GRBVar[] clause = new GRBVar[3];
  for (int j = 0; j < 3; j++) {
    if (Clauses[i,j] >= n) clause[j] = NotLit[Clauses[i,j]-n];
    else                   clause[j] = Lit[Clauses[i,j]];
  }
  model.AddGenConstrOr(Cla[i], clause, string.Format("CNSTR_Clause{0}", i));
}

// Link objs with clauses
model.AddGenConstrMin(Obj[0], Cla, GRB.INFINITY, "CNSTR_Obj0");
lhs = new GRBLinExpr();
for (i = 0; i < NCLAUSES; i++) {
  lhs.AddTerm(1.0, Cla[i]);
}
model.AddGenConstrIndicator(Obj[1], 1, lhs, GRB.GREATER_EQUAL, 4.0, "CNSTR_Obj1");

// Set global objective sense
model.ModelSense = GRB.MAXIMIZE;

// Save problem
model.Write("genconstr_cs.mps");
model.Write("genconstr_cs.lp");

// Optimize
model.Optimize();

// Status checking
status = model.Status;

if (status == GRB.Status.INF_OR_UNBD ||
    status == GRB.Status.INFEASIBLE  ||
    status == GRB.Status.UNBOUNDED     ) {
  Console.WriteLine("The model cannot be solved " +
        "because it is infeasible or unbounded");
  return;
}
if (status != GRB.Status.OPTIMAL) {
  Console.WriteLine("Optimization was stopped with status {0}", status);
  return;
}

// Print result
double objval = model.ObjVal;

if (objval > 1.9)
```

```
        Console.WriteLine("Logical expression is satisfiable");
      else if (objval > 0.9)
        Console.WriteLine("At least four clauses can be satisfied");
      else
        Console.WriteLine("Not even three clauses can be satisfied");

      // Dispose of model and environment
      model.Dispose();
      env.Dispose();

    } catch (GRBException e) {
      Console.WriteLine("Error code: {0}. {1}", e.ErrorCode, e.Message);
    }
  }
}
```

lp_cs.cs

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* This example reads an LP model from a file and solves it.
   If the model is infeasible or unbounded, the example turns off
   presolve and solves the model again. If the model is infeasible,
   the example computes an Irreducible Inconsistent Subsystem (IIS),
   and writes it to a file. */

using System;
using Gurobi;

class lp_cs
{
  static void Main(string[] args)
  {
    if (args.Length < 1) {
      Console.Out.WriteLine("Usage: lp_cs filename");
      return;
    }

    try {
      GRBEnv env = new GRBEnv();
      GRBModel model = new GRBModel(env, args[0]);

      model.Optimize();

      int optimstatus = model.Status;

      if (optimstatus == GRB.Status.INF_OR_UNBD) {
        model.Parameters.Presolve = 0;
        model.Optimize();
        optimstatus = model.Status;
      }

      if (optimstatus == GRB.Status.OPTIMAL) {
        double objval = model.ObjVal;
        Console.WriteLine("Optimal objective: " + objval);
      } else if (optimstatus == GRB.Status.INFEASIBLE) {
        Console.WriteLine("Model is infeasible");

        // compute and write out IIS

        model.ComputeIIS();
        model.Write("model.ilp");
```

```
      } else if (optimstatus == GRB.Status.UNBOUNDED) {
        Console.WriteLine("Model is unbounded");
      } else {
        Console.WriteLine("Optimization was stopped with status = "
                          + optimstatus);
      }

      // Dispose of model and env
      model.Dispose();
      env.Dispose();

    } catch (GRBException e) {
      Console.WriteLine("Error code: " + e.ErrorCode + ". " + e.Message);
    }
  }
}
```

lpmethod_cs.cs

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* Solve a model with different values of the Method parameter;
   show which value gives the shortest solve time. */

using System;
using Gurobi;

class lpmethod_cs
{
  static void Main(string[] args)
  {
    if (args.Length < 1) {
      Console.Out.WriteLine("Usage: lpmethod_cs filename");
      return;
    }

    try {
      // Read model
      GRBEnv env = new GRBEnv();
      GRBModel model = new GRBModel(env, args[0]);

      // Solve the model with different values of Method
      int bestMethod = -1;
      double bestTime = model.Parameters.TimeLimit;
      for (int i = 0; i <= 2; ++i)
      {
        model.Reset();
        model.Parameters.Method = i;
        model.Optimize();
        if (model.Status == GRB.Status.OPTIMAL)
        {
          bestTime = model.Runtime;
          bestMethod = i;
          // Reduce the TimeLimit parameter to save time
          // with other methods
          model.Parameters.TimeLimit = bestTime;
        }
      }

      // Report which method was fastest
      if (bestMethod == -1) {
        Console.WriteLine("Unable to solve this model");
      } else {
```

```
        Console.WriteLine("Solved in " + bestTime
          + " seconds with Method: " + bestMethod);
      }

      // Dispose of model and env
      model.Dispose();
      env.Dispose();

    } catch (GRBException e) {
      Console.WriteLine("Error code: " + e.ErrorCode + ". " + e.Message);
    }
  }
}
```

lpmod_cs.cs

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* This example reads an LP model from a file and solves it.
   If the model can be solved, then it finds the smallest positive variable,
   sets its upper bound to zero, and resolves the model two ways:
   first with an advanced start, then without an advanced start
   (i.e. 'from scratch'). */

using System;
using Gurobi;

class lpmod_cs
{
  static void Main(string[] args)
  {
    if (args.Length < 1) {
      Console.Out.WriteLine("Usage: lpmod_cs filename");
      return;
    }

    try {
      // Read model and determine whether it is an LP
      GRBEnv env = new GRBEnv();
      GRBModel model = new GRBModel(env, args[0]);
      if (model.IsMIP != 0) {
        Console.WriteLine("The model is not a linear program");
        Environment.Exit(1);
      }

      model.Optimize();

      int status = model.Status;

      if ((status == GRB.Status.INF_OR_UNBD) ||
          (status == GRB.Status.INFEASIBLE) ||
          (status == GRB.Status.UNBOUNDED)) {
        Console.WriteLine("The model cannot be solved because it is "
            + "infeasible or unbounded");
        Environment.Exit(1);
      }

      if (status != GRB.Status.OPTIMAL) {
        Console.WriteLine("Optimization was stopped with status " + status);
        Environment.Exit(0);
```

```
    }

    // Find the smallest variable value
    double minVal = GRB.INFINITY;
    GRBVar minVar = null;
    foreach (GRBVar v in model.GetVars()) {
      double sol = v.X;
      if ((sol > 0.0001) && (sol < minVal) && (v.LB == 0.0)) {
        minVal = sol;
        minVar = v;
      }
    }

    Console.WriteLine("\n*** Setting " +
        minVar.VarName + " from " + minVal +
        " to zero ***\n");
    minVar.UB = 0.0;

    // Solve from this starting point
    model.Optimize();

    // Save iteration & time info
    double warmCount = model.IterCount;
    double warmTime = model.Runtime;

    // Reset the model and resolve
    Console.WriteLine("\n*** Resetting and solving "
        + "without an advanced start ***\n");
    model.Reset();
    model.Optimize();

    double coldCount = model.IterCount;
    double coldTime = model.Runtime;

    Console.WriteLine("\n*** Warm start: " + warmCount + " iterations, " +
        warmTime + " seconds");
    Console.WriteLine("*** Cold start: " + coldCount + " iterations, " +
        coldTime + " seconds");

    // Dispose of model and env
    model.Dispose();
    env.Dispose();

  } catch (GRBException e) {
    Console.WriteLine("Error code: " + e.ErrorCode + ". " +
```

```
            e.Message);
    }
  }
}
```

mip1_cs.cs

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* This example formulates and solves the following simple MIP model:

     maximize    x +   y + 2 z
     subject to  x + 2 y + 3 z <= 4
                 x +   y       >= 1
     x, y, z binary
*/

using System;
using Gurobi;

class mip1_cs
{
  static void Main()
  {
    try {
      GRBEnv    env   = new GRBEnv("mip1.log");
      GRBModel  model = new GRBModel(env);

      // Create variables

      GRBVar x = model.AddVar(0.0, 1.0, 0.0, GRB.BINARY, "x");
      GRBVar y = model.AddVar(0.0, 1.0, 0.0, GRB.BINARY, "y");
      GRBVar z = model.AddVar(0.0, 1.0, 0.0, GRB.BINARY, "z");

      // Set objective: maximize x + y + 2 z

      model.SetObjective(x + y + 2 * z, GRB.MAXIMIZE);

      // Add constraint: x + 2 y + 3 z <= 4

      model.AddConstr(x + 2 * y + 3 * z <= 4.0, "c0");

      // Add constraint: x + y >= 1

      model.AddConstr(x + y >= 1.0, "c1");

      // Optimize model

      model.Optimize();

      Console.WriteLine(x.VarName + " " + x.X);
```

```
      Console.WriteLine(y.VarName + " " + y.X);
      Console.WriteLine(z.VarName + " " + z.X);

      Console.WriteLine("Obj: " + model.ObjVal);

      // Dispose of model and env

      model.Dispose();
      env.Dispose();

    } catch (GRBException e) {
      Console.WriteLine("Error code: " + e.ErrorCode + ". " + e.Message);
    }
  }
}
```

mip2_cs.cs

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* This example reads a MIP model from a file, solves it and
   prints the objective values from all feasible solutions
   generated while solving the MIP. Then it creates the fixed
   model and solves that model. */

using System;
using Gurobi;

class mip2_cs
{
  static void Main(string[] args)
  {
    if (args.Length < 1) {
      Console.Out.WriteLine("Usage: mip2_cs filename");
      return;
    }

    try {
      GRBEnv    env   = new GRBEnv();
      GRBModel  model = new GRBModel(env, args[0]);
      if (model.IsMIP == 0) {
        Console.WriteLine("Model is not a MIP");
        return;
      }

      model.Optimize();

      int optimstatus = model.Status;
      double objval = 0;
      if (optimstatus == GRB.Status.OPTIMAL) {
        objval = model.ObjVal;
        Console.WriteLine("Optimal objective: " + objval);
      } else if (optimstatus == GRB.Status.INF_OR_UNBD) {
        Console.WriteLine("Model is infeasible or unbounded");
        return;
      } else if (optimstatus == GRB.Status.INFEASIBLE) {
        Console.WriteLine("Model is infeasible");
        return;
      } else if (optimstatus == GRB.Status.UNBOUNDED) {
        Console.WriteLine("Model is unbounded");
        return;
      } else {
```

```
  Console.WriteLine("Optimization was stopped with status = "
                    + optimstatus);
  return;
}

/* Iterate over the solutions and compute the objectives */

GRBVar[] vars = model.GetVars();
model.Parameters.OutputFlag = 0;

Console.WriteLine();
for (int k = 0; k < model.SolCount; ++k) {
  model.Parameters.SolutionNumber = k;
  double objn = 0.0;

  for (int j = 0; j < vars.Length; j++) {
    objn += vars[j].Obj * vars[j].Xn;
  }

  Console.WriteLine("Solution " + k + " has objective: " + objn);
}
Console.WriteLine();
model.Parameters.OutputFlag = 1;

/* Create a fixed model, turn off presolve and solve */

GRBModel fixedmodel = model.FixedModel();

fixedmodel.Parameters.Presolve = 0;

fixedmodel.Optimize();

int foptimstatus = fixedmodel.Status;

if (foptimstatus != GRB.Status.OPTIMAL) {
  Console.WriteLine("Error: fixed model isn't optimal");
  return;
}

double fobjval = fixedmodel.ObjVal;

if (Math.Abs(fobjval - objval) > 1.0e-6 * (1.0 + Math.Abs(objval))) {
  Console.WriteLine("Error: objective values are different");
  return;
}
```

```
      GRBVar[] fvars  = fixedmodel.GetVars();
      double[] x      = fixedmodel.Get(GRB.DoubleAttr.X, fvars);
      string[] vnames = fixedmodel.Get(GRB.StringAttr.VarName, fvars);

      for (int j = 0; j < fvars.Length; j++) {
        if (x[j] != 0.0) Console.WriteLine(vnames[j] + " " + x[j]);
      }

      // Dispose of models and env
      fixedmodel.Dispose();
      model.Dispose();
      env.Dispose();

    } catch (GRBException e) {
      Console.WriteLine("Error code: " + e.ErrorCode + ". " + e.Message);
    }
  }
}
```

multiobj_cs.cs

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* Want to cover three different sets but subject to a common budget of
   elements allowed to be used. However, the sets have different priorities to
   be covered; and we tackle this by using multi-objective optimization. */

using System;
using Gurobi;

class multiobj_cs {
  static void Main() {

    try {
      // Sample data
      int groundSetSize = 20;
      int nSubsets      = 4;
      int Budget        = 12;
      double[,] Set = new double[,]
      { { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
        { 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1 },
        { 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0 },
        { 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0 } };
      int[]    SetObjPriority = new int[] {3, 2, 2, 1};
      double[] SetObjWeight   = new double[] {1.0, 0.25, 1.25, 1.0};
      int e, i, status, nSolutions;

      // Create environment
      GRBEnv env = new GRBEnv("multiobj_cs.log");

      // Create initial model
      GRBModel model = new GRBModel(env);
      model.ModelName = "multiobj_cs";

      // Initialize decision variables for ground set:
      // x[e] == 1 if element e is chosen for the covering.
      GRBVar[] Elem = model.AddVars(groundSetSize, GRB.BINARY);
      for (e = 0; e < groundSetSize; e++) {
        string vname = string.Format("El{0}", e);
        Elem[e].VarName = vname;
      }

      // Constraint: limit total number of elements to be picked to be at most
      // Budget
      GRBLinExpr lhs = new GRBLinExpr();
```

```
for (e = 0; e < groundSetSize; e++) {
  lhs.AddTerm(1.0, Elem[e]);
}
model.AddConstr(lhs, GRB.LESS_EQUAL, Budget, "Budget");

// Set global sense for ALL objectives
model.ModelSense = GRB.MAXIMIZE;

// Limit how many solutions to collect
model.Parameters.PoolSolutions = 100;

// Set and configure i-th objective
for (i = 0; i < nSubsets; i++) {
  string vname = string.Format("Set{0}", i);
  GRBLinExpr objn = new GRBLinExpr();
  for (e = 0; e < groundSetSize; e++) {
    objn.AddTerm(Set[i,e], Elem[e]);
  }

  model.SetObjectiveN(objn, i, SetObjPriority[i], SetObjWeight[i],
                      1.0 + i, 0.01, vname);
}

// Save problem
model.Write("multiobj_cs.lp");

// Optimize
model.Optimize();

// Status checking
status = model.Status;

if (status == GRB.Status.INF_OR_UNBD ||
    status == GRB.Status.INFEASIBLE  ||
    status == GRB.Status.UNBOUNDED     ) {
  Console.WriteLine("The model cannot be solved " +
           "because it is infeasible or unbounded");
  return;
}
if (status != GRB.Status.OPTIMAL) {
  Console.WriteLine("Optimization was stopped with status {0}", status);
  return;
}

// Print best selected set
```

```
      Console.WriteLine("Selected elements in best solution:");
      Console.Write("\t");
      for (e = 0; e < groundSetSize; e++) {
        if (Elem[e].X < .9) continue;
        Console.Write("El{0} ", e);
      }
      Console.WriteLine();

      // Print number of solutions stored
      nSolutions = model.SolCount;
      Console.WriteLine("Number of solutions found: {0}", nSolutions);

      // Print objective values of solutions
      if (nSolutions > 10) nSolutions = 10;
      Console.WriteLine("Objective values for first {0} solutions:", nSolutions);
      for (i = 0; i < nSubsets; i++) {
        model.Parameters.ObjNumber = i;

        Console.Write("\tSet" + i);
        for (e = 0; e < nSolutions; e++) {
          model.Parameters.SolutionNumber = e;
          Console.Write("{0,8}", model.ObjNVal);
        }
        Console.WriteLine();
      }
      model.Dispose();
      env.Dispose();
    } catch (GRBException e) {
      Console.WriteLine("Error code = {0}", e);
      Console.WriteLine(e.Message);
    }
  }
}
```

params_cs.cs

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* Use parameters that are associated with a model.

   A MIP is solved for a few seconds with different sets of parameters.
   The one with the smallest MIP gap is selected, and the optimization
   is resumed until the optimal solution is found.
*/

using System;
using Gurobi;

class params_cs
{
  static void Main(string[] args)
  {
    if (args.Length < 1) {
      Console.Out.WriteLine("Usage: params_cs filename");
      return;
    }

    try {
      // Read model and verify that it is a MIP
      GRBEnv env = new GRBEnv();
      GRBModel m = new GRBModel(env, args[0]);
      if (m.IsMIP == 0) {
        Console.WriteLine("The model is not an integer program");
        Environment.Exit(1);
      }

      // Set a 2 second time limit
      m.Parameters.TimeLimit = 2.0;

      // Now solve the model with different values of MIPFocus
      GRBModel bestModel = new GRBModel(m);
      bestModel.Optimize();
      for (int i = 1; i <= 3; ++i) {
        m.Reset();
        m.Parameters.MIPFocus = i;
        m.Optimize();
        if (bestModel.MIPGap > m.MIPGap) {
          GRBModel swap = bestModel;
          bestModel = m;
          m = swap;
```

```
        }
      }

      // Finally, delete the extra model, reset the time limit and
      // continue to solve the best model to optimality
      m.Dispose();
      bestModel.Parameters.TimeLimit = GRB.INFINITY;
      bestModel.Optimize();
      Console.WriteLine("Solved with MIPFocus: " +
                        bestModel.Parameters.MIPFocus);

    } catch (GRBException e) {
      Console.WriteLine("Error code: " + e.ErrorCode + ". " +
          e.Message);
    }
  }
}
```

piecewise_cs.cs

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* This example considers the following separable, convex problem:

     minimize    f(x) - y + g(z)
     subject to  x + 2 y + 3 z <= 4
                 x +   y       >= 1
                 x,    y,    z <= 1

   where f(u) = exp(-u) and g(u) = 2 u^2 - 4 u, for all real u. It
   formulates and solves a simpler LP model by approximating f and
   g with piecewise-linear functions. Then it transforms the model
   into a MIP by negating the approximation for f, which corresponds
   to a non-convex piecewise-linear function, and solves it again.
*/

using System;
using Gurobi;

class piecewise_cs
{

  private static double f(double u) { return Math.Exp(-u); }
  private static double g(double u) { return 2 * u * u - 4 * u; }

  static void Main()
  {
    try {

      // Create environment

      GRBEnv env = new GRBEnv();

      // Create a new model

      GRBModel model = new GRBModel(env);

      // Create variables

      double lb = 0.0, ub = 1.0;

      GRBVar x = model.AddVar(lb, ub, 0.0, GRB.CONTINUOUS, "x");
      GRBVar y = model.AddVar(lb, ub, 0.0, GRB.CONTINUOUS, "y");
      GRBVar z = model.AddVar(lb, ub, 0.0, GRB.CONTINUOUS, "z");
```

```
// Set objective for y

model.SetObjective(-y);

// Add piecewise-linear objective functions for x and z

int npts = 101;
double[] ptu = new double[npts];
double[] ptf = new double[npts];
double[] ptg = new double[npts];

for (int i = 0; i < npts; i++) {
  ptu[i] = lb + (ub - lb) * i / (npts - 1);
  ptf[i] = f(ptu[i]);
  ptg[i] = g(ptu[i]);
}

model.SetPWLObj(x, ptu, ptf);
model.SetPWLObj(z, ptu, ptg);

// Add constraint: x + 2 y + 3 z <= 4

model.AddConstr(x + 2 * y + 3 * z <= 4.0, "c0");

// Add constraint: x + y >= 1

model.AddConstr(x + y >= 1.0, "c1");

// Optimize model as an LP

model.Optimize();

Console.WriteLine("IsMIP: " + model.IsMIP);

Console.WriteLine(x.VarName + " " + x.X);
Console.WriteLine(y.VarName + " " + y.X);
Console.WriteLine(z.VarName + " " + z.X);

Console.WriteLine("Obj: " + model.ObjVal);

Console.WriteLine();

// Negate piecewise-linear objective function for x
```

```
      for (int i = 0; i < npts; i++) {
        ptf[i] = -ptf[i];
      }

      model.SetPWLObj(x, ptu, ptf);

      // Optimize model as a MIP

      model.Optimize();

      Console.WriteLine("IsMIP: " + model.IsMIP);

      Console.WriteLine(x.VarName + " " + x.X);
      Console.WriteLine(y.VarName + " " + y.X);
      Console.WriteLine(z.VarName + " " + z.X);

      Console.WriteLine("Obj: " + model.ObjVal);

      // Dispose of model and environment

      model.Dispose();
      env.Dispose();

    } catch (GRBException e) {
      Console.WriteLine("Error code: " + e.ErrorCode + ". " + e.Message);
    }
  }
}
```

poolsearch_cs.cs

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* We find alternative epsilon-optimal solutions to a given knapsack
   problem by using PoolSearchMode */

using System;
using Gurobi;

class poolsearch_cs {

  static void Main() {

    try{
      // Sample data
      int groundSetSize = 10;
      double[] objCoef = new double[] {32, 32, 15, 15, 6, 6, 1, 1, 1, 1};
      double[] knapsackCoef = new double[] {16, 16,  8,  8, 4, 4, 2, 2, 1, 1};
      double Budget = 33;
      int e, status, nSolutions;

      // Create environment
      GRBEnv env = new GRBEnv("poolsearch_cs.log");

      // Create initial model
      GRBModel model = new GRBModel(env);
      model.ModelName = "poolsearch_cs";

      // Initialize decision variables for ground set:
      // x[e] == 1 if element e is chosen
      GRBVar[] Elem = model.AddVars(groundSetSize, GRB.BINARY);
      model.Set(GRB.DoubleAttr.Obj, Elem, objCoef, 0, groundSetSize);

      for (e = 0; e < groundSetSize; e++) {
        Elem[e].VarName = string.Format("El{0}", e);
      }

      // Constraint: limit total number of elements to be picked to be at most
      // Budget
      GRBLinExpr lhs = new GRBLinExpr();
      for (e = 0; e < groundSetSize; e++) {
        lhs.AddTerm(knapsackCoef[e], Elem[e]);
      }
      model.AddConstr(lhs, GRB.LESS_EQUAL, Budget, "Budget");
```

```
// set global sense for ALL objectives
model.ModelSense = GRB.MAXIMIZE;

// Limit how many solutions to collect
model.Parameters.PoolSolutions = 1024;

// Limit the search space by setting a gap for the worst possible solution that will be ac
model.Parameters.PoolGap = 0.10;

// do a systematic search for the k-best solutions
model.Parameters.PoolSearchMode = 2;

// save problem
model.Write("poolsearch_cs.lp");

// Optimize
model.Optimize();

// Status checking
status = model.Status;

if (status == GRB.Status.INF_OR_UNBD ||
    status == GRB.Status.INFEASIBLE  ||
    status == GRB.Status.UNBOUNDED     ) {
  Console.WriteLine("The model cannot be solved " +
         "because it is infeasible or unbounded");
  return;
}
if (status != GRB.Status.OPTIMAL) {
  Console.WriteLine("Optimization was stopped with status {0}", status);
  return;
}

// Print best selected set
Console.WriteLine("Selected elements in best solution:");
Console.Write("\t");
for (e = 0; e < groundSetSize; e++) {
  if (Elem[e].X < .9) continue;
  Console.Write("El{0} ", e);
}
Console.WriteLine();

// Print number of solutions stored
nSolutions = model.SolCount;
Console.WriteLine("Number of solutions found: {0}", nSolutions);
```

```
      // Print objective values of solutions
      for (e = 0; e < nSolutions; e++) {
        model.Parameters.SolutionNumber = e;
        Console.Write("{0} ", model.PoolObjVal);
        if (e%15 == 14) Console.WriteLine();
      }
      Console.WriteLine();

      // Print fourth best set if available
      if (nSolutions >= 4) {
        model.Parameters.SolutionNumber = 3;

        Console.WriteLine("Selected elements in fourth best solution:");
        Console.Write("\t");
        for (e = 0; e < groundSetSize; e++) {
          if (Elem[e].Xn < .9) continue;
          Console.Write("El{0} ", e);
        }
        Console.WriteLine();
      }

      model.Dispose();
      env.Dispose();
    } catch (GRBException e) {
      Console.WriteLine("Error code: {0}. {1}", e.ErrorCode, e.Message);
    }
  }

}
```

qcp_cs.cs

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* This example formulates and solves the following simple QCP model:

     maximize    x
     subject to  x + y + z = 1
                 x^2 + y^2 <= z^2 (second-order cone)
                 x^2 <= yz        (rotated second-order cone)
*/

using System;
using Gurobi;

class qcp_cs
{
  static void Main()
  {
    try {
      GRBEnv    env   = new GRBEnv("qcp.log");
      GRBModel  model = new GRBModel(env);

      // Create variables

      GRBVar x = model.AddVar(0.0, GRB.INFINITY, 0.0, GRB.CONTINUOUS, "x");
      GRBVar y = model.AddVar(0.0, GRB.INFINITY, 0.0, GRB.CONTINUOUS, "y");
      GRBVar z = model.AddVar(0.0, GRB.INFINITY, 0.0, GRB.CONTINUOUS, "z");

      // Set objective

      GRBLinExpr obj = x;
      model.SetObjective(obj, GRB.MAXIMIZE);

      // Add linear constraint: x + y + z = 1

      model.AddConstr(x + y + z == 1.0, "c0");

      // Add second-order cone: x^2 + y^2 <= z^2

      model.AddQConstr(x*x + y*y <= z*z, "qc0");

      // Add rotated cone: x^2 <= yz

      model.AddQConstr(x*x <= y*z, "qc1");
```

```
      // Optimize model

      model.Optimize();

      Console.WriteLine(x.VarName + " " + x.X);
      Console.WriteLine(y.VarName + " " + y.X);
      Console.WriteLine(z.VarName + " " + z.X);

      Console.WriteLine("Obj: " + model.ObjVal + " " + obj.Value);

      // Dispose of model and env

      model.Dispose();
      env.Dispose();

    } catch (GRBException e) {
      Console.WriteLine("Error code: " + e.ErrorCode + ". " + e.Message);
    }
  }
}
```

qp_cs.cs

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* This example formulates and solves the following simple QP model:

     minimize    x^2 + x*y + y^2 + y*z + z^2 + 2 x
     subject to  x + 2 y + 3 z >= 4
                 x +   y       >= 1

   It solves it once as a continuous model, and once as an integer model.
*/

using System;
using Gurobi;

class qp_cs
{
  static void Main()
  {
    try {
      GRBEnv    env   = new GRBEnv("qp.log");
      GRBModel  model = new GRBModel(env);

      // Create variables

      GRBVar x = model.AddVar(0.0, 1.0, 0.0, GRB.CONTINUOUS, "x");
      GRBVar y = model.AddVar(0.0, 1.0, 0.0, GRB.CONTINUOUS, "y");
      GRBVar z = model.AddVar(0.0, 1.0, 0.0, GRB.CONTINUOUS, "z");

      // Set objective

      GRBQuadExpr obj = x*x + x*y + y*y + y*z + z*z + 2*x;
      model.SetObjective(obj);

      // Add constraint: x + 2 y + 3 z >= 4

      model.AddConstr(x + 2 * y + 3 * z >= 4.0, "c0");

      // Add constraint: x + y >= 1

      model.AddConstr(x + y >= 1.0, "c1");

      // Optimize model

      model.Optimize();
```

```
      Console.WriteLine(x.VarName + " " + x.X);
      Console.WriteLine(y.VarName + " " + y.X);
      Console.WriteLine(z.VarName + " " + z.X);

      Console.WriteLine("Obj: " + model.ObjVal + " " + obj.Value);


      // Change variable types to integer

      x.VType = GRB.INTEGER;
      y.VType = GRB.INTEGER;
      z.VType = GRB.INTEGER;

      // Optimize model

      model.Optimize();

      Console.WriteLine(x.VarName + " " + x.X);
      Console.WriteLine(y.VarName + " " + y.X);
      Console.WriteLine(z.VarName + " " + z.X);

      Console.WriteLine("Obj: " + model.ObjVal + " " + obj.Value);

      // Dispose of model and env

      model.Dispose();
      env.Dispose();

    } catch (GRBException e) {
      Console.WriteLine("Error code: " + e.ErrorCode + ". " + e.Message);
    }
  }
}
```

sensitivity_cs.cs

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* A simple sensitivity analysis example which reads a MIP model
   from a file and solves it. Then each binary variable is set
   to 1-X, where X is its value in the optimal solution, and
   the impact on the objective function value is reported.
*/

using System;
using Gurobi;

class sensitivity_cs
{
  static void Main(string[] args)
  {
    if (args.Length < 1) {
      Console.Out.WriteLine("Usage: sensitivity_cs filename");
      return;
    }

    try {

      // Create environment

      GRBEnv env = new GRBEnv();

      // Read and solve model

      GRBModel model = new GRBModel(env, args[0]);

      if (model.IsMIP == 0) {
        Console.WriteLine("Model is not a MIP");
        return;
      }

      model.Optimize();

      if (model.Status != GRB.Status.OPTIMAL) {
        Console.WriteLine("Optimization ended with status " + model.Status);
        return;
      }

      // Store the optimal solution
```

```
double   origObjVal = model.ObjVal;
GRBVar[] vars       = model.GetVars();
double[] origX      = model.Get(GRB.DoubleAttr.X, vars);

// Disable solver output for subsequent solves

model.Parameters.OutputFlag = 0;

// Iterate through unfixed, binary variables in model

for (int i = 0; i < vars.Length; i++) {
  GRBVar v     = vars[i];
  char   vType = v.VType;

  if (v.LB == 0 && v.UB == 1
      && (vType == GRB.BINARY || vType == GRB.INTEGER)) {

    // Set variable to 1-X, where X is its value in optimal solution

    if (origX[i] < 0.5) {
      v.LB = 1.0;
      v.Start = 1.0;
    } else {
      v.UB = 0.0;
      v.Start = 0.0;
    }

    // Update MIP start for the other variables

    for (int j = 0; j < vars.Length; j++) {
      if (j != i) {
        vars[j].Start = origX[j];
      }
    }

    // Solve for new value and capture sensitivity information

    model.Optimize();

    if (model.Status == GRB.Status.OPTIMAL) {
      Console.WriteLine("Objective sensitivity for variable "
          + v.VarName + " is " + (model.ObjVal - origObjVal));
    } else {
      Console.WriteLine("Objective sensitivity for variable "
          + v.VarName + " is infinite");
```

```
          }

          // Restore the original variable bounds

          v.LB = 0.0;
          v.UB = 1.0;
        }
      }

      // Dispose of model and environment

      model.Dispose();
      env.Dispose();

    } catch (GRBException e) {
      Console.WriteLine("Error code: " + e.ErrorCode);
      Console.WriteLine(e.Message);
      Console.WriteLine(e.StackTrace);
    }
  }
}
```

sos_cs.cs

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* This example creates a very simple Special Ordered Set (SOS) model.
   The model consists of 3 continuous variables, no linear constraints,
   and a pair of SOS constraints of type 1. */

using System;
using Gurobi;

class sos_cs
{
  static void Main()
  {
    try {
      GRBEnv env = new GRBEnv();

      GRBModel model = new GRBModel(env);

      // Create variables

      double[] ub    = {1, 1, 2};
      double[] obj   = {-2, -1, -1};
      string[] names = {"x0", "x1", "x2"};

      GRBVar[] x = model.AddVars(null, ub, obj, null, names);

      // Add first SOS1: x0=0 or x1=0

      GRBVar[] sosv1  = {x[0], x[1]};
      double[] soswt1 = {1, 2};

      model.AddSOS(sosv1, soswt1, GRB.SOS_TYPE1);

      // Add second SOS1: x0=0 or x2=0

      GRBVar[] sosv2  = {x[0], x[2]};
      double[] soswt2 = {1, 2};

      model.AddSOS(sosv2, soswt2, GRB.SOS_TYPE1);

      // Optimize model

      model.Optimize();
```

```
      for (int i = 0; i < 3; i++)
        Console.WriteLine(x[i].VarName + " " + x[i].X);

      // Dispose of model and env
      model.Dispose();
      env.Dispose();

    } catch (GRBException e) {
      Console.WriteLine("Error code: " + e.ErrorCode + ". " + e.Message);
    }
  }
}
```

sudoku_cs.cs

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/*
  Sudoku example.

  The Sudoku board is a 9x9 grid, which is further divided into a 3x3 grid
  of 3x3 grids.  Each cell in the grid must take a value from 0 to 9.
  No two grid cells in the same row, column, or 3x3 subgrid may take the
  same value.

  In the MIP formulation, binary variables x[i,j,v] indicate whether
  cell <i,j> takes value 'v'.  The constraints are as follows:
    1. Each cell must take exactly one value (sum_v x[i,j,v] = 1)
    2. Each value is used exactly once per row (sum_i x[i,j,v] = 1)
    3. Each value is used exactly once per column (sum_j x[i,j,v] = 1)
    4. Each value is used exactly once per 3x3 subgrid (sum_grid x[i,j,v] = 1)

  Input datasets for this example can be found in examples/data/sudoku*.
*/

using System;
using System.IO;
using Gurobi;

class sudoku_cs
{
  static void Main(string[] args)
  {
    int n = 9;
    int s = 3;

    if (args.Length < 1) {
      Console.Out.WriteLine("Usage: sudoku_cs filename");
      return;
    }

    try {
      GRBEnv env = new GRBEnv();
      GRBModel model = new GRBModel(env);

      // Create 3-D array of model variables

      GRBVar[,,] vars = new GRBVar[n,n,n];
```

```
for (int i = 0; i < n; i++) {
  for (int j = 0; j < n; j++) {
    for (int v = 0; v < n; v++) {
      string st = "G_" + i.ToString() + "_" + j.ToString()
                       + "_" + v.ToString();
      vars[i,j,v] = model.AddVar(0.0, 1.0, 0.0, GRB.BINARY, st);
    }
  }
}

// Add constraints

GRBLinExpr expr;

// Each cell must take one value

for (int i = 0; i < n; i++) {
  for (int j = 0; j < n; j++) {
    expr = 0.0;
    for (int v = 0; v < n; v++)
      expr.AddTerm(1.0, vars[i,j,v]);
    string st = "V_" + i.ToString() + "_" + j.ToString();
    model.AddConstr(expr == 1.0, st);
  }
}

// Each value appears once per row

for (int i = 0; i < n; i++) {
  for (int v = 0; v < n; v++) {
    expr = 0.0;
    for (int j = 0; j < n; j++)
      expr.AddTerm(1.0, vars[i,j,v]);
    string st = "R_" + i.ToString() + "_" + v.ToString();
    model.AddConstr(expr == 1.0, st);
  }
}

// Each value appears once per column

for (int j = 0; j < n; j++) {
  for (int v = 0; v < n; v++) {
    expr = 0.0;
    for (int i = 0; i < n; i++)
      expr.AddTerm(1.0, vars[i,j,v]);
```

```
    string st = "C_" + j.ToString() + "_" + v.ToString();
    model.AddConstr(expr == 1.0, st);
  }
}

// Each value appears once per sub-grid

for (int v = 0; v < n; v++) {
  for (int i0 = 0; i0 < s; i0++) {
    for (int j0 = 0; j0 < s; j0++) {
      expr = 0.0;
      for (int i1 = 0; i1 < s; i1++) {
        for (int j1 = 0; j1 < s; j1++) {
          expr.AddTerm(1.0, vars[i0*s+i1,j0*s+j1,v]);
        }
      }
      string st = "Sub_" + v.ToString() + "_" + i0.ToString()
                         + "_" + j0.ToString();
      model.AddConstr(expr == 1.0, st);
    }
  }
}

// Fix variables associated with pre-specified cells

StreamReader sr = File.OpenText(args[0]);

for (int i = 0; i < n; i++) {
  string input = sr.ReadLine();
  for (int j = 0; j < n; j++) {
    int val = (int) input[j] - 48 - 1; // 0-based

    if (val >= 0)
      vars[i,j,val].LB = 1.0;
  }
}

// Optimize model

model.Optimize();

// Write model to file
model.Write("sudoku.lp");

double[,,] x = model.Get(GRB.DoubleAttr.X, vars);
```

```
      Console.WriteLine();
      for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
          for (int v = 0; v < n; v++) {
            if (x[i,j,v] > 0.5) {
              Console.Write(v+1);
            }
          }
        }
        Console.WriteLine();
      }

      // Dispose of model and env
      model.Dispose();
      env.Dispose();

    } catch (GRBException e) {
      Console.WriteLine("Error code: " + e.ErrorCode + ". " + e.Message);
    }
  }
}
```

tsp_cs.cs

```
/* Copyright 2017, Gurobi Optimization, Inc. */

// Solve a traveling salesman problem on a randomly generated set of
// points using lazy constraints.   The base MIP model only includes
// 'degree-2' constraints, requiring each node to have exactly
// two incident edges.  Solutions to this model may contain subtours -
// tours that don't visit every node.  The lazy constraint callback
// adds new constraints to cut them off.

using System;
using Gurobi;

class tsp_cs : GRBCallback {
  private GRBVar[,] vars;

  public tsp_cs(GRBVar[,] xvars) {
    vars = xvars;
  }

  // Subtour elimination callback.  Whenever a feasible solution is found,
  // find the smallest subtour, and add a subtour elimination
  // constraint if the tour doesn't visit every node.

  protected override void Callback() {
    try {
      if (where == GRB.Callback.MIPSOL) {
        // Found an integer feasible solution - does it visit every node?

        int n = vars.GetLength(0);
        int[] tour = findsubtour(GetSolution(vars));

        if (tour.Length < n) {
          // Add subtour elimination constraint
          GRBLinExpr expr = 0;
          for (int i = 0; i < tour.Length; i++)
            for (int j = i+1; j < tour.Length; j++)
              expr.AddTerm(1.0, vars[tour[i], tour[j]]);
          AddLazy(expr <= tour.Length-1);
        }
      }
    } catch (GRBException e) {
      Console.WriteLine("Error code: " + e.ErrorCode + ". " + e.Message);
      Console.WriteLine(e.StackTrace);
    }
```

```
}

// Given an integer-feasible solution 'sol', return the smallest
// sub-tour (as a list of node indices).

protected static int[] findsubtour(double[,] sol)
{
  int n = sol.GetLength(0);
  bool[] seen = new bool[n];
  int[] tour = new int[n];
  int bestind, bestlen;
  int i, node, len, start;

  for (i = 0; i < n; i++)
    seen[i] = false;

  start = 0;
  bestlen = n+1;
  bestind = -1;
  node = 0;
  while (start < n) {
    for (node = 0; node < n; node++)
      if (!seen[node])
        break;
    if (node == n)
      break;
    for (len = 0; len < n; len++) {
      tour[start+len] = node;
      seen[node] = true;
      for (i = 0; i < n; i++) {
        if (sol[node, i] > 0.5 && !seen[i]) {
          node = i;
          break;
        }
      }
      if (i == n) {
        len++;
        if (len < bestlen) {
          bestlen = len;
          bestind = start;
        }
        start += len;
        break;
      }
    }
```

```
  }

  for (i = 0; i < bestlen; i++)
    tour[i] = tour[bestind+i];
  System.Array.Resize(ref tour, bestlen);

  return tour;
}

// Euclidean distance between points 'i' and 'j'

protected static double distance(double[] x,
                                 double[] y,
                                 int      i,
                                 int      j) {
  double dx = x[i]-x[j];
  double dy = y[i]-y[j];
  return Math.Sqrt(dx*dx+dy*dy);
}

public static void Main(String[] args) {

  if (args.Length < 1) {
    Console.WriteLine("Usage: tsp_cs nnodes");
    return;
  }

  int n = Convert.ToInt32(args[0]);

  try {
    GRBEnv   env   = new GRBEnv();
    GRBModel model = new GRBModel(env);

    // Must set LazyConstraints parameter when using lazy constraints

    model.Parameters.LazyConstraints = 1;

    double[] x = new double[n];
    double[] y = new double[n];

    Random r = new Random();
    for (int i = 0; i < n; i++) {
      x[i] = r.NextDouble();
      y[i] = r.NextDouble();
    }
```

```
    // Create variables

    GRBVar[,] vars = new GRBVar[n, n];

    for (int i = 0; i < n; i++) {
      for (int j = 0; j <= i; j++) {
        vars[i, j] = model.AddVar(0.0, 1.0, distance(x, y, i, j),
                                  GRB.BINARY, "x"+i+"_"+j);
        vars[j, i] = vars[i, j];
      }
    }

    // Degree-2 constraints

    for (int i = 0; i < n; i++) {
      GRBLinExpr expr = 0;
      for (int j = 0; j < n; j++)
        expr.AddTerm(1.0, vars[i, j]);
      model.AddConstr(expr == 2.0, "deg2_"+i);
    }

    // Forbid edge from node back to itself

    for (int i = 0; i < n; i++)
      vars[i, i].UB = 0.0;

    model.SetCallback(new tsp_cs(vars));
    model.Optimize();

    if (model.SolCount > 0) {
      int[] tour = findsubtour(model.Get(GRB.DoubleAttr.X, vars));

      Console.Write("Tour: ");
      for (int i = 0; i < tour.Length; i++)
        Console.Write(tour[i] + " ");
      Console.WriteLine();
    }

    // Dispose of model and environment
    model.Dispose();
    env.Dispose();

  } catch (GRBException e) {
    Console.WriteLine("Error code: " + e.ErrorCode + ". " + e.Message);
```

```
      Console.WriteLine(e.StackTrace);
    }
  }
}
```

tune_cs.cs

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* This example reads a model from a file and tunes it.
   It then writes the best parameter settings to a file
   and solves the model using these parameters. */

using System;
using Gurobi;

class tune_cs
{
  static void Main(string[] args)
  {
    if (args.Length < 1) {
      Console.Out.WriteLine("Usage: tune_cs filename");
      return;
    }

    try {
      GRBEnv env = new GRBEnv();

      // Read model from file
      GRBModel model = new GRBModel(env, args[0]);

      // Set the TuneResults parameter to 1
      model.Parameters.TuneResults = 1;

      // Tune the model
      model.Tune();

      // Get the number of tuning results
      int resultcount = model.TuneResultCount;

      if (resultcount > 0) {

        // Load the tuned parameters into the model's environment
        model.GetTuneResult(0);

        // Write the tuned parameters to a file
        model.Write("tune.prm");

        // Solve the model using the tuned parameters
        model.Optimize();
      }
```

```
      // Dispose of model and environment
      model.Dispose();
      env.Dispose();

    } catch (GRBException e) {
      Console.WriteLine("Error code: " + e.ErrorCode + ". " + e.Message);
    }
  }
}
```

workforce1_cs.cs

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* Assign workers to shifts; each worker may or may not be available on a
   particular day. If the problem cannot be solved, use IIS to find a set of
   conflicting constraints. Note that there may be additional conflicts
   besides what is reported via IIS. */

using System;
using Gurobi;

class workforce1_cs
{
  static void Main()
  {
    try {

      // Sample data
      // Sets of days and workers
      string[] Shifts =
          new string[] { "Mon1", "Tue2", "Wed3", "Thu4", "Fri5", "Sat6",
              "Sun7", "Mon8", "Tue9", "Wed10", "Thu11", "Fri12", "Sat13",
              "Sun14" };
      string[] Workers =
          new string[] { "Amy", "Bob", "Cathy", "Dan", "Ed", "Fred", "Gu" };

      int nShifts = Shifts.Length;
      int nWorkers = Workers.Length;

      // Number of workers required for each shift
      double[] shiftRequirements =
          new double[] { 3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5 };

      // Amount each worker is paid to work one shift
      double[] pay = new double[] { 10, 12, 10, 8, 8, 9, 11 };

      // Worker availability: 0 if the worker is unavailable for a shift
      double[,] availability =
          new double[,] { { 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1 },
              { 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0 },
              { 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
              { 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1 },
              { 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1 },
              { 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1 },
              { 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 } };
```

```
// Model
GRBEnv env = new GRBEnv();
GRBModel model = new GRBModel(env);

model.ModelName = "assignment";

// Assignment variables: x[w][s] == 1 if worker w is assigned
// to shift s. Since an assignment model always produces integer
// solutions, we use continuous variables and solve as an LP.
GRBVar[,] x = new GRBVar[nWorkers,nShifts];
for (int w = 0; w < nWorkers; ++w) {
  for (int s = 0; s < nShifts; ++s) {
    x[w,s] =
        model.AddVar(0, availability[w,s], pay[w], GRB.CONTINUOUS,
                     Workers[w] + "." + Shifts[s]);
  }
}

// The objective is to minimize the total pay costs
model.ModelSense = GRB.MINIMIZE;

// Constraint: assign exactly shiftRequirements[s] workers
// to each shift s
for (int s = 0; s < nShifts; ++s) {
  GRBLinExpr lhs = 0.0;
  for (int w = 0; w < nWorkers; ++w)
    lhs.AddTerm(1.0, x[w, s]);
  model.AddConstr(lhs == shiftRequirements[s], Shifts[s]);
}

// Optimize
model.Optimize();
int status = model.Status;
if (status == GRB.Status.UNBOUNDED) {
  Console.WriteLine("The model cannot be solved "
      + "because it is unbounded");
  return;
}
if (status == GRB.Status.OPTIMAL) {
  Console.WriteLine("The optimal objective is " + model.ObjVal);
  return;
}
if ((status != GRB.Status.INF_OR_UNBD) &&
    (status != GRB.Status.INFEASIBLE)) {
```

```
        Console.WriteLine("Optimization was stopped with status " + status);
        return;
      }

      // Do IIS
      Console.WriteLine("The model is infeasible; computing IIS");
      model.ComputeIIS();
      Console.WriteLine("\nThe following constraint(s) "
          + "cannot be satisfied:");
      foreach (GRBConstr c in model.GetConstrs()) {
        if (c.IISConstr == 1) {
          Console.WriteLine(c.ConstrName);
        }
      }

      // Dispose of model and env
      model.Dispose();
      env.Dispose();

    } catch (GRBException e) {
      Console.WriteLine("Error code: " + e.ErrorCode + ". " +
          e.Message);
    }
  }
}
```

workforce2_cs.cs

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* Assign workers to shifts; each worker may or may not be available on a
   particular day. If the problem cannot be solved, use IIS iteratively to
   find all conflicting constraints. */

using System;
using System.Collections.Generic;
using Gurobi;

class workforce2_cs
{
  static void Main()
  {
    try {

      // Sample data
      // Sets of days and workers
      string[] Shifts =
          new string[] { "Mon1", "Tue2", "Wed3", "Thu4", "Fri5", "Sat6",
              "Sun7", "Mon8", "Tue9", "Wed10", "Thu11", "Fri12", "Sat13",
              "Sun14" };
      string[] Workers =
          new string[] { "Amy", "Bob", "Cathy", "Dan", "Ed", "Fred", "Gu" };

      int nShifts = Shifts.Length;
      int nWorkers = Workers.Length;

      // Number of workers required for each shift
      double[] shiftRequirements =
          new double[] { 3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5 };

      // Amount each worker is paid to work one shift
      double[] pay = new double[] { 10, 12, 10, 8, 8, 9, 11 };

      // Worker availability: 0 if the worker is unavailable for a shift
      double[,] availability =
          new double[,] { { 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1 },
              { 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0 },
              { 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
              { 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1 },
              { 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1 },
              { 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1 },
              { 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 } };
```

```
// Model
GRBEnv env = new GRBEnv();
GRBModel model = new GRBModel(env);

model.ModelName = "assignment";

// Assignment variables: x[w][s] == 1 if worker w is assigned
// to shift s. Since an assignment model always produces integer
// solutions, we use continuous variables and solve as an LP.
GRBVar[,] x = new GRBVar[nWorkers,nShifts];
for (int w = 0; w < nWorkers; ++w) {
  for (int s = 0; s < nShifts; ++s) {
    x[w,s] =
        model.AddVar(0, availability[w,s], pay[w], GRB.CONTINUOUS,
                     Workers[w] + "." + Shifts[s]);
  }
}

// The objective is to minimize the total pay costs
model.ModelSense = GRB.MINIMIZE;

// Constraint: assign exactly shiftRequirements[s] workers
// to each shift s
for (int s = 0; s < nShifts; ++s) {
  GRBLinExpr lhs = 0.0;
  for (int w = 0; w < nWorkers; ++w)
    lhs.AddTerm(1.0, x[w, s]);
  model.AddConstr(lhs == shiftRequirements[s], Shifts[s]);
}

// Optimize
model.Optimize();
int status = model.Status;
if (status == GRB.Status.UNBOUNDED) {
  Console.WriteLine("The model cannot be solved "
      + "because it is unbounded");
  return;
}
if (status == GRB.Status.OPTIMAL) {
  Console.WriteLine("The optimal objective is " + model.ObjVal);
  return;
}
if ((status != GRB.Status.INF_OR_UNBD) &&
    (status != GRB.Status.INFEASIBLE)) {
```

```
  Console.WriteLine("Optimization was stopped with status " + status);
  return;
}

// Do IIS
Console.WriteLine("The model is infeasible; computing IIS");
LinkedList<string> removed = new LinkedList<string>();

// Loop until we reduce to a model that can be solved
while (true) {
  model.ComputeIIS();
  Console.WriteLine("\nThe following constraint cannot be satisfied:");
  foreach (GRBConstr c in model.GetConstrs()) {
    if (c.IISConstr == 1) {
      Console.WriteLine(c.ConstrName);
      // Remove a single constraint from the model
      removed.AddFirst(c.ConstrName);
      model.Remove(c);
      break;
    }
  }

  Console.WriteLine();
  model.Optimize();
  status = model.Status;

  if (status == GRB.Status.UNBOUNDED) {
    Console.WriteLine("The model cannot be solved "
        + "because it is unbounded");
    return;
  }
  if (status == GRB.Status.OPTIMAL) {
    break;
  }
  if ((status != GRB.Status.INF_OR_UNBD) &&
      (status != GRB.Status.INFEASIBLE)) {
    Console.WriteLine("Optimization was stopped with status " +
        status);
    return;
  }
}

Console.WriteLine("\nThe following constraints were removed "
    + "to get a feasible LP:");
foreach (string s in removed) {
```

```
        Console.Write(s + " ");
      }
      Console.WriteLine();

      // Dispose of model and env
      model.Dispose();
      env.Dispose();

    } catch (GRBException e) {
      Console.WriteLine("Error code: " + e.ErrorCode + ". " +
          e.Message);
    }
  }
}
```

workforce3_cs.cs

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* Assign workers to shifts; each worker may or may not be available on a
   particular day. If the problem cannot be solved, relax the model
   to determine which constraints cannot be satisfied, and how much
   they need to be relaxed. */

using System;
using Gurobi;

class workforce3_cs
{
  static void Main()
  {
    try {

      // Sample data
      // Sets of days and workers
      string[] Shifts =
          new string[] { "Mon1", "Tue2", "Wed3", "Thu4", "Fri5", "Sat6",
              "Sun7", "Mon8", "Tue9", "Wed10", "Thu11", "Fri12", "Sat13",
              "Sun14" };
      string[] Workers =
          new string[] { "Amy", "Bob", "Cathy", "Dan", "Ed", "Fred", "Gu" };

      int nShifts = Shifts.Length;
      int nWorkers = Workers.Length;

      // Number of workers required for each shift
      double[] shiftRequirements =
          new double[] { 3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5 };

      // Amount each worker is paid to work one shift
      double[] pay = new double[] { 10, 12, 10, 8, 8, 9, 11 };

      // Worker availability: 0 if the worker is unavailable for a shift
      double[,] availability =
          new double[,] { { 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1 },
              { 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0 },
              { 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
              { 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1 },
              { 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1 },
              { 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1 },
              { 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 } };
```

```
// Model
GRBEnv env = new GRBEnv();
GRBModel model = new GRBModel(env);

model.ModelName = "assignment";

// Assignment variables: x[w][s] == 1 if worker w is assigned
// to shift s. Since an assignment model always produces integer
// solutions, we use continuous variables and solve as an LP.
GRBVar[,] x = new GRBVar[nWorkers,nShifts];
for (int w = 0; w < nWorkers; ++w) {
  for (int s = 0; s < nShifts; ++s) {
    x[w,s] =
        model.AddVar(0, availability[w,s], pay[w], GRB.CONTINUOUS,
                     Workers[w] + "." + Shifts[s]);
  }
}

// The objective is to minimize the total pay costs
model.ModelSense = GRB.MINIMIZE;

// Constraint: assign exactly shiftRequirements[s] workers
// to each shift s
for (int s = 0; s < nShifts; ++s) {
  GRBLinExpr lhs = 0.0;
  for (int w = 0; w < nWorkers; ++w) {
    lhs.AddTerm(1.0, x[w,s]);
  }
  model.AddConstr(lhs == shiftRequirements[s], Shifts[s]);
}

// Optimize
model.Optimize();
int status = model.Status;
if (status == GRB.Status.UNBOUNDED) {
  Console.WriteLine("The model cannot be solved "
      + "because it is unbounded");
  return;
}
if (status == GRB.Status.OPTIMAL) {
  Console.WriteLine("The optimal objective is " + model.ObjVal);
  return;
}
if ((status != GRB.Status.INF_OR_UNBD) &&
```

```
          (status != GRB.Status.INFEASIBLE)) {
        Console.WriteLine("Optimization was stopped with status " + status);
        return;
      }

      // Relax the constraints to make the model feasible
      Console.WriteLine("The model is infeasible; relaxing the constraints");
      int orignumvars = model.NumVars;
      model.FeasRelax(0, false, false, true);
      model.Optimize();
      status = model.Status;
      if ((status == GRB.Status.INF_OR_UNBD) ||
          (status == GRB.Status.INFEASIBLE) ||
          (status == GRB.Status.UNBOUNDED)) {
        Console.WriteLine("The relaxed model cannot be solved "
            + "because it is infeasible or unbounded");
        return;
      }
      if (status != GRB.Status.OPTIMAL) {
        Console.WriteLine("Optimization was stopped with status " + status);
        return;
      }

      Console.WriteLine("\nSlack values:");
      GRBVar[] vars = model.GetVars();
      for (int i = orignumvars; i < model.NumVars; ++i) {
        GRBVar sv = vars[i];
        if (sv.X > 1e-6) {
          Console.WriteLine(sv.VarName + " = " + sv.X);
        }
      }

      // Dispose of model and environment
      model.Dispose();
      env.Dispose();

    } catch (GRBException e) {
      Console.WriteLine("Error code: " + e.ErrorCode + ". " +
          e.Message);
    }
  }
}
```

workforce4_cs.cs

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* Assign workers to shifts; each worker may or may not be available on a
   particular day. We use Pareto optimization to solve the model:
   first, we minimize the linear sum of the slacks. Then, we constrain
   the sum of the slacks, and we minimize a quadratic objective that
   tries to balance the workload among the workers. */

using System;
using Gurobi;

class workforce4_cs
{
  static void Main()
  {
    try {

      // Sample data
      // Sets of days and workers
      string[] Shifts =
          new string[] { "Mon1", "Tue2", "Wed3", "Thu4", "Fri5", "Sat6",
              "Sun7", "Mon8", "Tue9", "Wed10", "Thu11", "Fri12", "Sat13",
              "Sun14" };
      string[] Workers =
          new string[] { "Amy", "Bob", "Cathy", "Dan", "Ed", "Fred", "Gu" };

      int nShifts = Shifts.Length;
      int nWorkers = Workers.Length;

      // Number of workers required for each shift
      double[] shiftRequirements =
          new double[] { 3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5 };

      // Worker availability: 0 if the worker is unavailable for a shift
      double[,] availability =
          new double[,] { { 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1 },
              { 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0 },
              { 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
              { 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1 },
              { 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1 },
              { 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1 },
              { 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 } };

      // Model
```

```
GRBEnv env = new GRBEnv();
GRBModel model = new GRBModel(env);

model.ModelName = "assignment";

// Assignment variables: x[w][s] == 1 if worker w is assigned
// to shift s. This is no longer a pure assignment model, so we must
// use binary variables.
GRBVar[,] x = new GRBVar[nWorkers, nShifts];
for (int w = 0; w < nWorkers; ++w) {
  for (int s = 0; s < nShifts; ++s) {
    x[w,s] =
        model.AddVar(0, availability[w,s], 0, GRB.BINARY,
                     Workers[w] + "." + Shifts[s]);
  }
}

// Slack variables for each shift constraint so that the shifts can
// be satisfied
GRBVar[] slacks = new GRBVar[nShifts];
for (int s = 0; s < nShifts; ++s) {
  slacks[s] =
    model.AddVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS,
                 Shifts[s] + "Slack");
}

// Variable to represent the total slack
GRBVar totSlack = model.AddVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS,
                               "totSlack");

// Variables to count the total shifts worked by each worker
GRBVar[] totShifts = new GRBVar[nWorkers];
for (int w = 0; w < nWorkers; ++w) {
  totShifts[w] = model.AddVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS,
                              Workers[w] + "TotShifts");
}

GRBLinExpr lhs;

// Constraint: assign exactly shiftRequirements[s] workers
// to each shift s, plus the slack
for (int s = 0; s < nShifts; ++s) {
  lhs = new GRBLinExpr();
  lhs.AddTerm(1.0, slacks[s]);
  for (int w = 0; w < nWorkers; ++w) {
```

```
    lhs.AddTerm(1.0, x[w, s]);
  }
  model.AddConstr(lhs == shiftRequirements[s], Shifts[s]);
}

// Constraint: set totSlack equal to the total slack
lhs = new GRBLinExpr();
for (int s = 0; s < nShifts; ++s) {
  lhs.AddTerm(1.0, slacks[s]);
}
model.AddConstr(lhs == totSlack, "totSlack");

// Constraint: compute the total number of shifts for each worker
for (int w = 0; w < nWorkers; ++w) {
  lhs = new GRBLinExpr();
  for (int s = 0; s < nShifts; ++s) {
    lhs.AddTerm(1.0, x[w, s]);
  }
  model.AddConstr(lhs == totShifts[w], "totShifts" + Workers[w]);
}

// Objective: minimize the total slack
model.SetObjective(1.0*totSlack);

// Optimize
int status = solveAndPrint(model, totSlack, nWorkers, Workers, totShifts);
if (status != GRB.Status.OPTIMAL) {
  return;
}

// Constrain the slack by setting its upper and lower bounds
totSlack.UB = totSlack.X;
totSlack.LB = totSlack.X;

// Variable to count the average number of shifts worked
GRBVar avgShifts =
  model.AddVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS, "avgShifts");

// Variables to count the difference from average for each worker;
// note that these variables can take negative values.
GRBVar[] diffShifts = new GRBVar[nWorkers];
for (int w = 0; w < nWorkers; ++w) {
  diffShifts[w] = model.AddVar(-GRB.INFINITY, GRB.INFINITY, 0,
                               GRB.CONTINUOUS, Workers[w] + "Diff");
}
```

```
    // Constraint: compute the average number of shifts worked
    lhs = new GRBLinExpr();
    for (int w = 0; w < nWorkers; ++w) {
      lhs.AddTerm(1.0, totShifts[w]);
    }
    model.AddConstr(lhs == nWorkers * avgShifts, "avgShifts");

    // Constraint: compute the difference from the average number of shifts
    for (int w = 0; w < nWorkers; ++w) {
      model.AddConstr(totShifts[w] - avgShifts == diffShifts[w],
                      Workers[w] + "Diff");
    }

    // Objective: minimize the sum of the square of the difference from the
    // average number of shifts worked
    GRBQuadExpr qobj = new GRBQuadExpr();
    for (int w = 0; w < nWorkers; ++w) {
      qobj.AddTerm(1.0, diffShifts[w], diffShifts[w]);
    }
    model.SetObjective(qobj);

    // Optimize
    status = solveAndPrint(model, totSlack, nWorkers, Workers, totShifts);
    if (status != GRB.Status.OPTIMAL) {
      return;
    }

    // Dispose of model and env
    model.Dispose();
    env.Dispose();

  } catch (GRBException e) {
    Console.WriteLine("Error code: " + e.ErrorCode + ". " +
        e.Message);
  }
}

private static int solveAndPrint(GRBModel model, GRBVar totSlack,
                                 int nWorkers, String[] Workers,
                                 GRBVar[] totShifts)
{
  model.Optimize();
  int status = model.Status;
  if ((status == GRB.Status.INF_OR_UNBD) ||
```

```
        (status == GRB.Status.INFEASIBLE) ||
        (status == GRB.Status.UNBOUNDED)) {
      Console.WriteLine("The model cannot be solved "
          + "because it is infeasible or unbounded");
      return status;
    }
    if (status != GRB.Status.OPTIMAL) {
      Console.WriteLine("Optimization was stopped with status " + status);
      return status;
    }

    // Print total slack and the number of shifts worked for each worker
    Console.WriteLine("\nTotal slack required: " + totSlack.X);
    for (int w = 0; w < nWorkers; ++w) {
      Console.WriteLine(Workers[w] + " worked " +
                        totShifts[w].X + " shifts");
    }
    Console.WriteLine("\n");
    return status;
  }
}
```

workforce5_cs.cs

```
/* Copyright 2017, Gurobi Optimization, Inc. */

/* Assign workers to shifts; each worker may or may not be available on a
   particular day. We use multi-objective optimization to solve the model.
   The highest-priority objective minimizes the sum of the slacks
   (i.e., the total number of uncovered shifts). The secondary objective
   minimizes the difference between the maximum and minimum number of
   shifts worked among all workers.  The second optimization is allowed
   to degrade the first objective by up to the smaller value of 10% and 2 */

using System;
using Gurobi;

class workforce5_cs
{
  static void Main()
  {
    try {

      // Sample data
      // Sets of days and workers
      string[] Shifts =
          new string[]  { "Mon1", "Tue2", "Wed3", "Thu4", "Fri5", "Sat6",
              "Sun7", "Mon8", "Tue9", "Wed10", "Thu11", "Fri12", "Sat13",
              "Sun14" };
      string[] Workers =
          new string[] { "Amy", "Bob", "Cathy", "Dan", "Ed", "Fred", "Gu", "Tobi" };

      int nShifts = Shifts.Length;
      int nWorkers = Workers.Length;

      // Number of workers required for each shift
      double[] shiftRequirements =
          new double[] { 3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5 };

      // Worker availability: 0 if the worker is unavailable for a shift
      double[,] availability =
          new double[,] { { 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1 },
              { 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0 },
              { 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
              { 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1 },
              { 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1 },
              { 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1 },
              { 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1 },
```

```
        { 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 } };

// Create environment
GRBEnv env = new GRBEnv();

// Create initial model
GRBModel model = new GRBModel(env);
model.ModelName = "workforce5_cs";

// Initialize assignment decision variables:
// x[w][s] == 1 if worker w is assigned to shift s.
// This is no longer a pure assignment model, so we must
// use binary variables.
GRBVar[,] x = new GRBVar[nWorkers, nShifts];
for (int w = 0; w < nWorkers; ++w) {
  for (int s = 0; s < nShifts; ++s) {
    x[w,s] =
        model.AddVar(0, availability[w,s], 0, GRB.BINARY,
                     string.Format("{0}.{1}", Workers[w], Shifts[s]));
  }
}

// Slack variables for each shift constraint so that the shifts can
// be satisfied
GRBVar[] slacks = new GRBVar[nShifts];
for (int s = 0; s < nShifts; ++s) {
  slacks[s] =
      model.AddVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS,
                   string.Format("{0}Slack", Shifts[s]));
}

// Variable to represent the total slack
GRBVar totSlack = model.AddVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS,
                               "totSlack");

// Variables to count the total shifts worked by each worker
GRBVar[] totShifts = new GRBVar[nWorkers];
for (int w = 0; w < nWorkers; ++w) {
  totShifts[w] = model.AddVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS,
                              string.Format("{0}TotShifts", Workers[w]));
}

GRBLinExpr lhs;

// Constraint: assign exactly shiftRequirements[s] workers
```

```
// to each shift s, plus the slack
for (int s = 0; s < nShifts; ++s) {
  lhs = new GRBLinExpr();
  lhs.AddTerm(1.0, slacks[s]);
  for (int w = 0; w < nWorkers; ++w) {
    lhs.AddTerm(1.0, x[w,s]);
  }
  model.AddConstr(lhs, GRB.EQUAL, shiftRequirements[s], Shifts[s]);
}

// Constraint: set totSlack equal to the total slack
lhs = new GRBLinExpr();
lhs.AddTerm(-1.0, totSlack);
for (int s = 0; s < nShifts; ++s) {
  lhs.AddTerm(1.0, slacks[s]);
}
model.AddConstr(lhs, GRB.EQUAL, 0, "totSlack");

// Constraint: compute the total number of shifts for each worker
for (int w = 0; w < nWorkers; ++w) {
  lhs = new GRBLinExpr();
  lhs.AddTerm(-1.0, totShifts[w]);
  for (int s = 0; s < nShifts; ++s) {
    lhs.AddTerm(1.0, x[w,s]);
  }
  model.AddConstr(lhs, GRB.EQUAL, 0, string.Format("totShifts{0}", Workers[w]));
}

// Constraint: set minShift/maxShift variable to less <=/>= to the
// number of shifts among all workers
GRBVar minShift = model.AddVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS,
                               "minShift");
GRBVar maxShift = model.AddVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS,
                               "maxShift");
model.AddGenConstrMin(minShift, totShifts, GRB.INFINITY, "minShift");
model.AddGenConstrMax(maxShift, totShifts, -GRB.INFINITY, "maxShift");

// Set global sense for ALL objectives
model.ModelSense = GRB.MINIMIZE;

// Set primary objective
model.SetObjectiveN(totSlack, 0, 2, 1.0, 2.0, 0.1, "TotalSlack");

// Set secondary objective
model.SetObjectiveN(maxShift - minShift, 1, 1, 1.0, 0, 0, "Fairness");
```

```
    // Save problem
    model.Write("workforce5_cs.lp");

    // Optimize
    int status = solveAndPrint(model, totSlack, nWorkers, Workers, totShifts);

    if (status != GRB.Status.OPTIMAL)
      return;

    // Dispose of model and environment
    model.Dispose();
    env.Dispose();

  } catch (GRBException e) {
    Console.WriteLine("Error code: {0}. {1}", e.ErrorCode, e.Message);
  }
}

private static int solveAndPrint(GRBModel model, GRBVar totSlack,
                                 int nWorkers, String[] Workers,
                                 GRBVar[] totShifts)
{

  model.Optimize();
  int status = model.Status;
  if (status == GRB.Status.INF_OR_UNBD ||
      status == GRB.Status.INFEASIBLE  ||
      status == GRB.Status.UNBOUNDED     ) {
    Console.WriteLine("The model cannot be solved "
        + "because it is infeasible or unbounded");
    return status;
  }
  if (status != GRB.Status.OPTIMAL ) {
    Console.WriteLine("Optimization was stopped with status {0}", status);
    return status;
  }

  // Print total slack and the number of shifts worked for each worker
  Console.WriteLine("\nTotal slack required: {0}", totSlack.X);
  for (int w = 0; w < nWorkers; ++w) {
    Console.WriteLine("{0} worked {1} shifts", Workers[w], totShifts[w].X);
  }
  Console.WriteLine("\n");
  return status;
```

```
  }

}
```

3.5 Visual Basic Examples

This section includes source code for all of the Gurobi Visual Basic examples. The same source code can be found in the `examples/vb` directory of the Gurobi distribution.

callback_vb.vb

```
' Copyright 2017, Gurobi Optimization, Inc.

'   This example reads a model from a file, sets up a callback that
'   monitors optimization progress and implements a custom
'   termination strategy, and outputs progress information to the
'   screen and to a log file.
'
'   The termination strategy implemented in this callback stops the
'   optimization of a MIP model once at least one of the following two
'   conditions have been satisfied:
'     1) The optimality gap is less than 10%
'     2) At least 10000 nodes have been explored, and an integer feasible
'        solution has been found.
'   Note that termination is normally handled through Gurobi parameters
'   (MIPGap, NodeLimit, etc.).  You should only use a callback for
'   termination if the available parameters don't capture your desired
'   termination criterion.

Imports System
Imports Gurobi

Class callback_vb
    Inherits GRBCallback
    Private vars As GRBVar()
    Private lastnode As Double
    Private lastiter As Double

    Public Sub New(ByVal xvars As GRBVar())
        vars = xvars
        lastnode = lastiter = -1
    End Sub

    Protected Overloads Overrides Sub Callback()
        Try
            If where = GRB.Callback.PRESOLVE Then
                ' Presolve callback
                Dim cdels As Integer = GetIntInfo(GRB.Callback.PRE_COLDEL)
                Dim rdels As Integer = GetIntInfo(GRB.Callback.PRE_ROWDEL)
                Console.WriteLine(cdels & " columns and " & rdels & " rows are removed")
```

```
        ElseIf where = GRB.Callback.SIMPLEX Then
            ' Simplex callback
            Dim itcnt As Double = GetDoubleInfo(GRB.Callback.SPX_ITRCNT)
            If itcnt Mod - lastiter >= 100 Then
                lastiter = itcnt
                Dim obj As Double = GetDoubleInfo(GRB.Callback.SPX_OBJVAL)
                Dim pinf As Double = GetDoubleInfo(GRB.Callback.SPX_PRIMINF)
                Dim dinf As Double = GetDoubleInfo(GRB.Callback.SPX_DUALINF)
                Dim ispert As Integer = GetIntInfo(GRB.Callback.SPX_ISPERT)
                Dim ch As Char
                If ispert = 0 Then
                    ch = " "c
                ElseIf ispert = 1 Then
                    ch = "S"c
                Else
                    ch = "P"c
                End If
                Console.WriteLine(itcnt & "  " & obj & ch & "  " & pinf & "  " & dinf)
            End If
        ElseIf where = GRB.Callback.MIP Then
            ' General MIP callback
            Dim nodecnt As Double = GetDoubleInfo(GRB.Callback.MIP_NODCNT)
            If nodecnt - lastnode >= 100 Then
                lastnode = nodecnt
                Dim objbst As Double = GetDoubleInfo(GRB.Callback.MIP_OBJBST)
                Dim objbnd As Double = GetDoubleInfo(GRB.Callback.MIP_OBJBND)
                If Math.Abs(objbst - objbnd) < 0.1 * (1.0R + Math.Abs(objbst)) Then
                    Abort()
                End If
                Dim actnodes As Integer = CInt(GetDoubleInfo(GRB.Callback.MIP_NODLFT))
                Dim itcnt As Integer = CInt(GetDoubleInfo(GRB.Callback.MIP_ITRCNT))
                Dim solcnt As Integer = GetIntInfo(GRB.Callback.MIP_SOLCNT)
                Dim cutcnt As Integer = GetIntInfo(GRB.Callback.MIP_CUTCNT)
                Console.WriteLine(nodecnt & " " & actnodes & " " & itcnt & " " & _
                                  objbst & " " & objbnd & " " & solcnt & " " & cutcnt)
            End If
        ElseIf where = GRB.Callback.MIPSOL Then
            ' MIP solution callback
            Dim obj As Double = GetDoubleInfo(GRB.Callback.MIPSOL_OBJ)
            Dim nodecnt As Integer = CInt(GetDoubleInfo(GRB.Callback.MIPSOL_NODCNT))
            Dim x As Double() = GetSolution(vars)
            Console.WriteLine("**** New solution at node " & nodecnt & ", obj " & _
                              obj & ", x(0) = " & x(0) & "****")
        End If
    Catch e As GRBException
```

```
            Console.WriteLine("Error code: " & e.ErrorCode & ". " & e.Message)
            Console.WriteLine(e.StackTrace)
        End Try
    End Sub

    Shared Sub Main(ByVal args As String())

        If args.Length < 1 Then
            Console.WriteLine("Usage: callback_vb filename")
            Return
        End If

        Try
            Dim env As New GRBEnv()
            Dim model As New GRBModel(env, args(0))

            Dim vars As GRBVar() = model.GetVars()

            ' Create a callback object and associate it with the model
            model.SetCallback(New callback_vb(vars))
            model.Optimize()

            Dim x As Double() = model.Get(GRB.DoubleAttr.X, vars)
            Dim vnames As String() = model.Get(GRB.StringAttr.VarName, vars)

            For j As Integer = 0 To vars.Length - 1
                If x(j) <> 0.0R Then
                    Console.WriteLine(vnames(j) & " " & x(j))
                End If
            Next

            ' Dispose of model and env
            model.Dispose()
            env.Dispose()

        Catch e As GRBException
            Console.WriteLine("Error code: " & e.ErrorCode & ". " & e.Message)
            Console.WriteLine(e.StackTrace)
        End Try
    End Sub
End Class
```

dense_vb.vb

```
' Copyright 2017, Gurobi Optimization, Inc.
'
' This example formulates and solves the following simple QP model:
'
'   minimize    x + y + x^2 + x*y + y^2 + y*z + z^2
'   subject to  x + 2 y + 3 z >= 4
'               x +   y       >= 1
'
' The example illustrates the use of dense matrices to store A and Q
' (and dense vectors for the other relevant data).  We don't recommend
' that you use dense matrices, but this example may be helpful if you
' already have your data in this format.

Imports Gurobi

Class dense_vb

    Protected Shared Function _
      dense_optimize(env As GRBEnv, _
                     rows As Integer, _
                     cols As Integer, _
                     c As Double(), _
                     Q As Double(,), _
                     A As Double(,), _
                     sense As Char(), _
                     rhs As Double(), _
                     lb As Double(), _
                     ub As Double(), _
                     vtype As Char(), _
                     solution As Double()) As Boolean

        Dim success As Boolean = False

        Try
            Dim model As New GRBModel(env)

            ' Add variables to the model

            Dim vars As GRBVar() = model.AddVars(lb, ub, Nothing, vtype, Nothing)

            ' Populate A matrix

            For i As Integer = 0 To rows - 1
                Dim expr As New GRBLinExpr()
```

```
            For j As Integer = 0 To cols - 1
                If A(i, j) <> 0 Then
                    expr.AddTerm(A(i, j), vars(j))
                End If
            Next
            model.AddConstr(expr, sense(i), rhs(i), "")
        Next

        ' Populate objective

        Dim obj As New GRBQuadExpr()
        If Q IsNot Nothing Then
            For i As Integer = 0 To cols - 1
                For j As Integer = 0 To cols - 1
                    If Q(i, j) <> 0 Then
                        obj.AddTerm(Q(i, j), vars(i), vars(j))
                    End If
                Next
            Next
            For j As Integer = 0 To cols - 1
                If c(j) <> 0 Then
                    obj.AddTerm(c(j), vars(j))
                End If
            Next
            model.SetObjective(obj)
        End If

        ' Solve model

        model.Optimize()

        ' Extract solution

        If model.Status = GRB.Status.OPTIMAL Then
            success = True

            For j As Integer = 0 To cols - 1
                solution(j) = vars(j).X
            Next
        End If

        model.Dispose()

    Catch e As GRBException
        Console.WriteLine("Error code: " & e.ErrorCode & ". " & e.Message)
```

```
        End Try

        Return success
    End Function

    Public Shared Sub Main(args As String())
        Try
            Dim env As New GRBEnv()

            Dim c As Double() = New Double() {1, 1, 0}
            Dim Q As Double(,) = New Double(,) {{1, 1, 0}, {0, 1, 1}, {0, 0, 1}}
            Dim A As Double(,) = New Double(,) {{1, 2, 3}, {1, 1, 0}}
            Dim sense As Char() = New Char() {">"C, ">"C}
            Dim rhs As Double() = New Double() {4, 1}
            Dim lb As Double() = New Double() {0, 0, 0}
            Dim success As Boolean
            Dim sol As Double() = New Double(2) {}

            success = dense_optimize(env, 2, 3, c, Q, A, sense, rhs, lb, Nothing, _
                                     Nothing, sol)

            If success Then
                Console.WriteLine("x: " & sol(0) & ", y: " & sol(1) & ", z: " & sol(2))
            End If

            ' Dispose of environment

            env.Dispose()

        Catch e As GRBException
            Console.WriteLine("Error code: " & e.ErrorCode & ". " & e.Message)
        End Try

    End Sub
End Class
```

diet_vb.vb

```
' Copyright 2017, Gurobi Optimization, Inc.

' Solve the classic diet model, showing how to add constraints
' to an existing model.

Imports System
Imports Gurobi

Class diet_vb
    Shared Sub Main()
        Try

            ' Nutrition guidelines, based on
            ' USDA Dietary Guidelines for Americans, 2005
            ' http://www.health.gov/DietaryGuidelines/dga2005/
            Dim Categories As String() = New String() {"calories", "protein", "fat", _
                                                       "sodium"}
            Dim nCategories As Integer = Categories.Length
            Dim minNutrition As Double() = New Double() {1800, 91, 0, 0}
            Dim maxNutrition As Double() = New Double() {2200, GRB.INFINITY, 65, 1779}

            ' Set of foods
            Dim Foods As String() = New String() {"hamburger", "chicken", "hot dog", _
                                                  "fries", "macaroni", "pizza", _
                                                  "salad", "milk", "ice cream"}
            Dim nFoods As Integer = Foods.Length
            Dim cost As Double() = New Double() {2.49, 2.89, 1.5R, 1.89, 2.09, 1.99, _
                                                 2.49, 0.89, 1.59}

            ' Nutrition values for the foods
            ' hamburger
            ' chicken
            ' hot dog
            ' fries
            ' macaroni
            ' pizza
            ' salad
            ' milk
            ' ice cream
            Dim nutritionValues As Double(,) = New Double(,) {{410, 24, 26, 730}, _
                                                              {420, 32, 10, 1190}, _
                                                              {560, 20, 32, 1800}, _
                                                              {380, 4, 19, 270}, _
                                                              {320, 12, 10, 930}, _
```

```
                                                    {320, 15, 12, 820}, _
                                                    {320, 31, 12, 1230}, _
                                                    {100, 8, 2.5, 125}, _
                                                    {330, 8, 10, 180}}

' Model
Dim env As New GRBEnv()
Dim model As New GRBModel(env)

model.ModelName = "diet"

' Create decision variables for the nutrition information,
' which we limit via bounds
Dim nutrition As GRBVar() = New GRBVar(nCategories - 1) {}
For i As Integer = 0 To nCategories - 1
    nutrition(i) = model.AddVar(minNutrition(i), maxNutrition(i), 0, _
                                GRB.CONTINUOUS, Categories(i))
Next

' Create decision variables for the foods to buy
Dim buy As GRBVar() = New GRBVar(nFoods - 1) {}
For j As Integer = 0 To nFoods - 1
    buy(j) = model.AddVar(0, GRB.INFINITY, cost(j), GRB.CONTINUOUS, _
                          Foods(j))
Next

' The objective is to minimize the costs
model.ModelSense = GRB.MINIMIZE

' Nutrition constraints
For i As Integer = 0 To nCategories - 1
    Dim ntot As GRBLinExpr = 0
    For j As Integer = 0 To nFoods - 1
        ntot.AddTerm(nutritionValues(j, i), buy(j))
    Next
    model.AddConstr(ntot = nutrition(i), Categories(i))
Next

' Solve
model.Optimize()
PrintSolution(model, buy, nutrition)

Console.WriteLine(vbLf & "Adding constraint: at most 6 servings of dairy")
model.AddConstr(buy(7) + buy(8) <= 6, "limit_dairy")
```

```
            ' Solve
            model.Optimize()

            PrintSolution(model, buy, nutrition)

            ' Dispose of model and env
            model.Dispose()
            env.Dispose()

        Catch e As GRBException
            Console.WriteLine("Error code: " & e.ErrorCode & ". " & e.Message)
        End Try
    End Sub

    Private Shared Sub PrintSolution(ByVal model As GRBModel, ByVal buy As GRBVar(), _
                                     ByVal nutrition As GRBVar())
        If model.Status = GRB.Status.OPTIMAL Then
            Console.WriteLine(vbLf & "Cost: " & model.ObjVal)
            Console.WriteLine(vbLf & "Buy:")
            For j As Integer = 0 To buy.Length - 1
                If buy(j).X > 0.0001 Then
                    Console.WriteLine(buy(j).VarName & " " & buy(j).X)
                End If
            Next
            Console.WriteLine(vbLf & "Nutrition:")
            For i As Integer = 0 To nutrition.Length - 1
                Console.WriteLine(nutrition(i).VarName & " " & nutrition(i).X)
            Next
        Else
            Console.WriteLine("No solution")
        End If
    End Sub
End Class
```

facility_vb.vb

```
' Copyright 2017, Gurobi Optimization, Inc.
'
' Facility location: a company currently ships its product from 5 plants
' to 4 warehouses. It is considering closing some plants to reduce
' costs. What plant(s) should the company close, in order to minimize
' transportation and fixed costs?
'
' Based on an example from Frontline Systems:
' http://www.solver.com/disfacility.htm
' Used with permission.

Imports System
Imports Gurobi

Class facility_vb
    Shared Sub Main()
        Try

            ' Warehouse demand in thousands of units
            Dim Demand As Double() = New Double() {15, 18, 14, 20}

            ' Plant capacity in thousands of units
            Dim Capacity As Double() = New Double() {20, 22, 17, 19, 18}

            ' Fixed costs for each plant
            Dim FixedCosts As Double() = New Double() {12000, 15000, 17000, 13000, _
                                                       16000}

            ' Transportation costs per thousand units
            Dim TransCosts As Double(,) = New Double(,) {{4000, 2000, 3000, 2500, 4500}, _
                                                         {2500, 2600, 3400, 3000, 4000}, _
                                                         {1200, 1800, 2600, 4100, 3000}, _
                                                         {2200, 2600, 3100, 3700, 3200}}

            ' Number of plants and warehouses
            Dim nPlants As Integer = Capacity.Length
            Dim nWarehouses As Integer = Demand.Length

            ' Model
            Dim env As New GRBEnv()
            Dim model As New GRBModel(env)

            model.ModelName = "facility"
```

```
' Plant open decision variables: open(p) == 1 if plant p is open.
Dim open As GRBVar() = New GRBVar(nPlants - 1) {}
For p As Integer = 0 To nPlants - 1
    open(p) = model.AddVar(0, 1, FixedCosts(p), GRB.BINARY, "Open" & p)
Next

' Transportation decision variables: how much to transport from
' a plant p to a warehouse w
Dim transport As GRBVar(,) = New GRBVar(nWarehouses - 1, nPlants - 1) {}
For w As Integer = 0 To nWarehouses - 1
    For p As Integer = 0 To nPlants - 1
        transport(w, p) = model.AddVar(0, GRB.INFINITY, _
                                       TransCosts(w, p), GRB.CONTINUOUS, _
                                       "Trans" & p & "." & w)
    Next
Next

' The objective is to minimize the total fixed and variable costs
model.ModelSense = GRB.MINIMIZE

' Production constraints
' Note that the right-hand limit sets the production to zero if
' the plant is closed
For p As Integer = 0 To nPlants - 1
    Dim ptot As GRBLinExpr = 0
    For w As Integer = 0 To nWarehouses - 1
        ptot.AddTerm(1.0, transport(w, p))
    Next
    model.AddConstr(ptot <= Capacity(p) * open(p), "Capacity" & p)
Next

' Demand constraints
For w As Integer = 0 To nWarehouses - 1
    Dim dtot As GRBLinExpr = 0
    For p As Integer = 0 To nPlants - 1
        dtot.AddTerm(1.0, transport(w, p))
    Next
    model.AddConstr(dtot = Demand(w), "Demand" & w)
Next

' Guess at the starting point: close the plant with the highest
' fixed costs; open all others

' First, open all plants
For p As Integer = 0 To nPlants - 1
```

```
    open(p).Start = 1.0
Next

' Now close the plant with the highest fixed cost
Console.WriteLine("Initial guess:")
Dim maxFixed As Double = -GRB.INFINITY
For p As Integer = 0 To nPlants - 1
    If FixedCosts(p) > maxFixed Then
        maxFixed = FixedCosts(p)
    End If
Next
For p As Integer = 0 To nPlants - 1
    If FixedCosts(p) = maxFixed Then
        open(p).Start = 0.0
        Console.WriteLine("Closing plant " & p & vbLf)
        Exit For
    End If
Next

' Use barrier to solve root relaxation
model.Parameters.Method = GRB.METHOD_BARRIER

' Solve
model.Optimize()

' Print solution
Console.WriteLine(vbLf & "TOTAL COSTS: " & model.ObjVal)
Console.WriteLine("SOLUTION:")
For p As Integer = 0 To nPlants - 1
    If open(p).X > 0.99 Then
        Console.WriteLine("Plant " & p & " open:")
        For w As Integer = 0 To nWarehouses - 1
            If transport(w, p).X > 0.0001 Then
                Console.WriteLine("  Transport " & _
                                  transport(w, p).X & _
                                  " units to warehouse " & w)
            End If
        Next
    Else
        Console.WriteLine("Plant " & p & " closed!")
    End If

Next

' Dispose of model and env
```

```
            model.Dispose()
            env.Dispose()

        Catch e As GRBException
            Console.WriteLine("Error code: " & e.ErrorCode & ". " & e.Message)
        End Try
    End Sub
End Class
```

feasopt_vb.vb

```
' Copyright 2017, Gurobi Optimization, Inc.
'
' This example reads a MIP model from a file, adds artificial
' variables to each constraint, and then minimizes the sum of the
' artificial variables.  A solution with objective zero corresponds
' to a feasible solution to the input model.
' We can also use FeasRelax feature to do it. In this example, we
' use minrelax=1, i.e. optimizing the returned model finds a solution
' that minimizes the original objective, but only from among those
' solutions that minimize the sum of the artificial variables.

Imports Gurobi
Imports System

Class feasopt_vb
    Shared Sub Main(ByVal args As String())

        If args.Length < 1 Then
            Console.WriteLine("Usage: feasopt_vb filename")
            Return
        End If

        Try
            Dim env As New GRBEnv()
            Dim feasmodel As New GRBModel(env, args(0))

            'Create a copy to use FeasRelax feature later
            Dim feasmodel1 As New GRBModel(feasmodel)

            ' Clear objective
            feasmodel.SetObjective(New GRBLinExpr())

            ' Add slack variables
            Dim c As GRBConstr() = feasmodel.GetConstrs()
            For i As Integer = 0 To c.Length - 1
                Dim sense As Char = c(i).Sense
                If sense <> ">"c Then
                    Dim constrs As GRBConstr() = New GRBConstr() {c(i)}
                    Dim coeffs As Double() = New Double() {-1}
                    feasmodel.AddVar(0.0, GRB.INFINITY, 1.0, GRB.CONTINUOUS, _
                                     constrs, coeffs, _
                                     "ArtN_" & c(i).ConstrName)
                End If
                If sense <> "<"c Then
```

```
                Dim constrs As GRBConstr() = New GRBConstr() {c(i)}
                Dim coeffs As Double() = New Double() {1}
                feasmodel.AddVar(0.0, GRB.INFINITY, 1.0, GRB.CONTINUOUS, _
                                 constrs, coeffs, _
                                 "ArtP_" & c(i).ConstrName)
            End If
        Next

        ' Optimize modified model
        feasmodel.Optimize()
        feasmodel.Write("feasopt.lp")

        ' Use FeasRelax feature */
        feasmodel1.FeasRelax(GRB.FEASRELAX_LINEAR, true, false, true)
        feasmodel1.Write("feasopt1.lp")
        feasmodel1.Optimize()

        ' Dispose of model and env
        feasmodel1.Dispose()
        feasmodel.Dispose()
        env.Dispose()

    Catch e As GRBException
        Console.WriteLine("Error code: " & e.ErrorCode & ". " & e.Message)
    End Try
  End Sub
End Class
```

fixanddive_vb.vb

```
' Copyright 2017, Gurobi Optimization, Inc.
'
' Implement a simple MIP heuristic.  Relax the model,
' sort variables based on fractionality, and fix the 25% of
' the fractional variables that are closest to integer variables.
' Repeat until either the relaxation is integer feasible or
' linearly infeasible.

Imports System
Imports System.Collections.Generic
Imports Gurobi

Class fixanddive_vb
    ' Comparison class used to sort variable list based on relaxation
    ' fractionality

    Private Class FractionalCompare : Implements IComparer(Of GRBVar)
        Public Function Compare(ByVal v1 As GRBVar, ByVal v2 As GRBVar) As Integer _
                            Implements IComparer(Of Gurobi.GRBVar).Compare
            Try
                Dim sol1 As Double = Math.Abs(v1.X)
                Dim sol2 As Double = Math.Abs(v2.X)
                Dim frac1 As Double = Math.Abs(sol1 - Math.Floor(sol1 + 0.5))
                Dim frac2 As Double = Math.Abs(sol2 - Math.Floor(sol2 + 0.5))
                If frac1 < frac2 Then
                    Return -1
                ElseIf frac1 > frac2 Then
                    Return 1
                Else
                    Return 0
                End If
            Catch e As GRBException
                Console.WriteLine("Error code: " & e.ErrorCode & ". " & e.Message)
            End Try
            Return 0
        End Function
    End Class

    Shared Sub Main(ByVal args As String())

        If args.Length < 1 Then
            Console.WriteLine("Usage: fixanddive_vb filename")
            Return
        End If
```

```
Try
    ' Read model
    Dim env As New GRBEnv()
    Dim model As New GRBModel(env, args(0))

    ' Collect integer variables and relax them
    Dim intvars As New List(Of GRBVar)()
    For Each v As GRBVar In model.GetVars()
        If v.VType <> GRB.CONTINUOUS Then
            intvars.Add(v)
            v.VType = GRB.CONTINUOUS
        End If
    Next

    model.Parameters.OutputFlag = 0
    model.Optimize()

    ' Perform multiple iterations. In each iteration, identify the first
    ' quartile of integer variables that are closest to an integer value
    ' in the relaxation, fix them to the nearest integer, and repeat.

    For iter As Integer = 0 To 999

        ' create a list of fractional variables, sorted in order of
        ' increasing distance from the relaxation solution to the nearest
        ' integer value

        Dim fractional As New List(Of GRBVar)()
        For Each v As GRBVar In intvars
            Dim sol As Double = Math.Abs(v.X)
            If Math.Abs(sol - Math.Floor(sol + 0.5)) > 0.00001 Then
                fractional.Add(v)
            End If
        Next

        Console.WriteLine("Iteration " & iter & ", obj " & _
                          model.ObjVal & ", fractional " & fractional.Count)

        If fractional.Count = 0 Then
            Console.WriteLine("Found feasible solution - objective " & _
                              model.ObjVal)
            Exit For
        End If
```

```
                ' Fix the first quartile to the nearest integer value

                fractional.Sort(New FractionalCompare())
                Dim nfix As Integer = Math.Max(fractional.Count / 4, 1)
                For i As Integer = 0 To nfix - 1
                    Dim v As GRBVar = fractional(i)
                    Dim fixval As Double = Math.Floor(v.X + 0.5)
                    v.LB = fixval
                    v.UB = fixval
                    Console.WriteLine("  Fix " & v.VarName & " to " & fixval & _
                                      " ( rel " & v.X & " )")
                Next

                model.Optimize()

                ' Check optimization result

                If model.Status <> GRB.Status.OPTIMAL Then
                    Console.WriteLine("Relaxation is infeasible")
                    Exit For
                End If
            Next

            ' Dispose of model and env
            model.Dispose()
            env.Dispose()

        Catch e As GRBException
            Console.WriteLine("Error code: " & e.ErrorCode & ". " + e.Message)
        End Try
    End Sub
End Class
```

genconstr_vb.vb

```
' Copyright 2017, Gurobi Optimization, Inc.

' In this example we show the use of general constraints for modeling
' some common expressions. We use as an example a SAT-problem where we
' want to see if it is possible to satisfy at least four (or all) clauses
' of the logical for
'
' L = (x0 or ~x1 or x2)  and (x1 or ~x2 or x3)  and
'     (x2 or ~x3 or x0)  and (x3 or ~x0 or x1)  and
'     (~x0 or ~x1 or x2) and (~x1 or ~x2 or x3) and
'     (~x2 or ~x3 or x0) and (~x3 or ~x0 or x1)
'
' We do this by introducing two variables for each literal (itself and its
' negated value), a variable for each clause, and then two
' variables for indicating if we can satisfy four, and another to identify
' the minimum of the clauses (so if it one, we can satisfy all clauses)
' and put these two variables in the objective.
' i.e. the Objective function will be
'
' maximize Obj0 + Obj1
'
'  Obj0 = MIN(Clause1, ... , Clause8)
'  Obj1 = 1 -> Clause1 + ... + Clause8 >= 4
'
' thus, the objective value will be two if and only if we can satisfy all
' clauses; one if and only of at least four clauses can be satisfied, and
' zero otherwise.
'

Imports Gurobi

Class genconstr_vb

    Public Const n As Integer = 4
    Public Const NLITERALS As Integer = 4  'same as n
    Public Const NCLAUSES As Integer = 8
    Public Const NOBJ As Integer = 2

    Shared Sub Main()

        Try

            ' Example data:
```

```
' e.g. {0, n+1, 2} means clause (x0 or ~x1 or x2)
Dim Clauses As Integer(,) = New Integer(,) { _
    {    0, n + 1, 2}, {    1, n + 2, 3}, _
    {    2, n + 3, 0}, {    3, n + 0, 1}, _
    {n + 0, n + 1, 2}, {n + 1, n + 2, 3}, _
    {n + 2, n + 3, 0}, {n + 3, n + 0, 1}}

Dim i As Integer, status As Integer

' Create environment
Dim env As New GRBEnv("genconstr_vb.log")

' Create initial model
Dim model As New GRBModel(env)
model.ModelName = "genconstr_vb"

' Initialize decision variables and objective
Dim Lit As GRBVar() = New GRBVar(NLITERALS - 1) {}
Dim NotLit As GRBVar() = New GRBVar(NLITERALS - 1) {}
For i = 0 To NLITERALS - 1
    Lit(i) = model.AddVar(0.0, 1.0, 0.0, GRB.BINARY, String.Format("X{0}", i))
    NotLit(i) = model.AddVar(0.0, 1.0, 0.0, GRB.BINARY, String.Format("notX{0}", i))
Next

Dim Cla As GRBVar() = New GRBVar(NCLAUSES - 1) {}
For i = 0 To NCLAUSES - 1
    Cla(i) = model.AddVar(0.0, 1.0, 0.0, GRB.BINARY, String.Format("Clause{0}", i))
Next

Dim Obj As GRBVar() = New GRBVar(NOBJ - 1) {}
For i = 0 To NOBJ - 1
    Obj(i) = model.AddVar(0.0, 1.0, 1.0, GRB.BINARY, String.Format("Obj{0}", i))
Next

' Link Xi and notXi
Dim lhs As GRBLinExpr
For i = 0 To NLITERALS - 1
    lhs = New GRBLinExpr()
    lhs.AddTerm(1.0, Lit(i))
    lhs.AddTerm(1.0, NotLit(i))
    model.AddConstr(lhs, GRB.EQUAL, 1.0, String.Format("CNSTR_X{0}", i))
Next

' Link clauses and literals
For i = 0 To NCLAUSES - 1
```

```
    Dim clause As GRBVar() = New GRBVar(2) {}
    For j As Integer = 0 To 2
        If Clauses(i, j) >= n Then
            clause(j) = NotLit(Clauses(i, j) - n)
        Else
            clause(j) = Lit(Clauses(i, j))
        End If
    Next
    model.AddGenConstrOr(Cla(i), clause, String.Format("CNSTR_Clause{0}", i))
Next

' Link objs with clauses
model.AddGenConstrMin(Obj(0), Cla, GRB.INFINITY, "CNSTR_Obj0")
lhs = New GRBLinExpr()
For i = 0 To NCLAUSES - 1
    lhs.AddTerm(1.0, Cla(i))
Next
model.AddGenConstrIndicator(Obj(1), 1, lhs, GRB.GREATER_EQUAL, 4.0, "CNSTR_Obj1")

' Set global objective sense
model.ModelSense = GRB.MAXIMIZE

' Save problem
model.Write("genconstr_vb.mps")
model.Write("genconstr_vb.lp")

' Optimize
model.Optimize()

' Status checking
status = model.Status

If status = GRB.Status.INF_OR_UNBD OrElse _
   status = GRB.Status.INFEASIBLE OrElse _
   status = GRB.Status.UNBOUNDED Then
    Console.WriteLine("The model cannot be solved " & _
             "because it is infeasible or unbounded")
    Return
End If

If status <> GRB.Status.OPTIMAL Then
    Console.WriteLine("Optimization was stopped with status {0}", status)
    Return
End If
```

```
            ' Print result
            Dim objval As Double = model.ObjVal

            If objval > 1.9 Then
                Console.WriteLine("Logical expression is satisfiable")
            ElseIf objval > 0.9 Then
                Console.WriteLine("At least four clauses can be satisfied")
            Else
                Console.WriteLine("Not even three clauses can be satisfied")
            End If

            ' Dispose of model and environment
            model.Dispose()
            env.Dispose()

        Catch e As GRBException
            Console.WriteLine("Error code: {0}. {1}", e.ErrorCode, e.Message)

        End Try
    End Sub
End Class
```

lp_vb.vb

```
' Copyright 2017, Gurobi Optimization, Inc.
'
' This example reads an LP model from a file and solves it.
' If the model is infeasible or unbounded, the example turns off
' presolve and solves the model again. If the model is infeasible,
' the example computes an Irreducible Inconsistent Subsystem (IIS),
' and writes it to a file.

Imports System
Imports Gurobi

Class lp_vb
    Shared Sub Main(ByVal args As String())

        If args.Length < 1 Then
            Console.WriteLine("Usage: lp_vb filename")
            Return
        End If

        Try
            Dim env As GRBEnv = New GRBEnv("lp1.log")
            Dim model As GRBModel = New GRBModel(env, args(0))

            model.Optimize()

            Dim optimstatus As Integer = model.Status

            If optimstatus = GRB.Status.INF_OR_UNBD Then
                model.Parameters.Presolve = 0
                model.Optimize()
                optimstatus = model.Status
            End If

            If optimstatus = GRB.Status.OPTIMAL Then
                Dim objval As Double = model.ObjVal
                Console.WriteLine("Optimal objective: " & objval)
            ElseIf optimstatus = GRB.Status.INFEASIBLE Then
                Console.WriteLine("Model is infeasible")
                model.ComputeIIS()
                model.Write("model.ilp")
            ElseIf optimstatus = GRB.Status.UNBOUNDED Then
                Console.WriteLine("Model is unbounded")
            Else
                Console.WriteLine("Optimization was stopped with status = " & _
```

```
                                optimstatus)
            End If

            ' Dispose of model and env
            model.Dispose()
            env.Dispose()

        Catch e As GRBException
            Console.WriteLine("Error code: " & e.ErrorCode & ". " & e.Message)
        End Try
    End Sub
End Class
```

lpmethod_vb.vb

```
' Copyright 2017, Gurobi Optimization, Inc.
'
' Solve a model with different values of the Method parameter;
' show which value gives the shortest solve time.

Imports System
Imports Gurobi

Class lpmethod_vb

    Shared Sub Main(ByVal args As String())

        If args.Length < 1 Then
            Console.WriteLine("Usage: lpmethod_vb filename")
            Return
        End If

        Try
            ' Read model and verify that it is a MIP
            Dim env As New GRBEnv()
            Dim model As New GRBModel(env, args(0))

            ' Solve the model with different values of Method
            Dim bestMethod As Integer = -1
            Dim bestTime As Double = model.get(GRB.DoubleParam.TimeLimit)
            For i As Integer = 0 To 2
                model.Reset()
                model.Parameters.Method = i
                model.Optimize()
                If model.Status = GRB.Status.OPTIMAL Then
                    bestTime = model.Runtime
                    bestMethod = i
                    ' Reduce the TimeLimit parameter to save time
                    ' with other methods
                    model.Parameters.TimeLimit = bestTime
                End If
            Next

            ' Report which method was fastest
            If bestMethod = -1 Then
                Console.WriteLine("Unable to solve this model")
            Else
                Console.WriteLine("Solved in " & bestTime & _
                                  " seconds with Method: " & bestMethod)
```

```
            End If

            ' Dispose of model and env
            model.Dispose()
            env.Dispose()

        Catch e As GRBException
            Console.WriteLine("Error code: " & e.ErrorCode & ". " & e.Message)
        End Try
    End Sub
End Class
```

lpmod_vb.vb

```
' Copyright 2017, Gurobi Optimization, Inc.
'
' This example reads an LP model from a file and solves it.
' If the model can be solved, then it finds the smallest positive variable,
' sets its upper bound to zero, and resolves the model two ways:
' first with an advanced start, then without an advanced start
' (i.e. from scratch).

Imports System
Imports Gurobi

Class lpmod_vb
    Shared Sub Main(ByVal args As String())

        If args.Length < 1 Then
            Console.WriteLine("Usage: lpmod_vb filename")
            Return
        End If

        Try
            ' Read model and determine whether it is an LP
            Dim env As New GRBEnv()
            Dim model As New GRBModel(env, args(0))
            If model.IsMIP <> 0 Then
                Console.WriteLine("The model is not a linear program")
                Environment.Exit(1)
            End If

            model.Optimize()

            Dim status As Integer = model.Status

            If (status = GRB.Status.INF_OR_UNBD) OrElse _
               (status = GRB.Status.INFEASIBLE) OrElse _
               (status = GRB.Status.UNBOUNDED) Then
                Console.WriteLine("The model cannot be solved because it is " & _
                                  "infeasible or unbounded")
                Environment.Exit(1)
            End If

            If status <> GRB.Status.OPTIMAL Then
                Console.WriteLine("Optimization was stopped with status " & status)
                Environment.Exit(0)
            End If
```

```
        ' Find the smallest variable value
        Dim minVal As Double = GRB.INFINITY
        Dim minVar As GRBVar = Nothing
        For Each v As GRBVar In model.GetVars()
            Dim sol As Double = v.X
            If (sol > 0.0001) AndAlso _
               (sol < minVal) AndAlso _
               (v.LB = 0.0) Then
                minVal = sol
                minVar = v
            End If
        Next

        Console.WriteLine(vbLf & "*** Setting " & _
                          minVar.VarName & " from " & minVal & " to zero ***" & vbLf)
        minVar.UB = 0

        ' Solve from this starting point
        model.Optimize()

        ' Save iteration & time info
        Dim warmCount As Double = model.IterCount
        Dim warmTime As Double = model.Runtime

        ' Reset the model and resolve
        Console.WriteLine(vbLf & "*** Resetting and solving " & _
                          "without an advanced start ***" & vbLf)
        model.Reset()
        model.Optimize()

        Dim coldCount As Double = model.IterCount
        Dim coldTime As Double = model.Runtime

        Console.WriteLine(vbLf & "*** Warm start: " & warmCount & _
                          " iterations, " & warmTime & " seconds")

        Console.WriteLine("*** Cold start: " & coldCount & " iterations, " & _
                          coldTime & " seconds")

        ' Dispose of model and env
        model.Dispose()
        env.Dispose()

    Catch e As GRBException
```

```
            Console.WriteLine("Error code: " & e.ErrorCode & ". " & e.Message)
        End Try
    End Sub
End Class
```

mip1_vb.vb

```
' Copyright 2017, Gurobi Optimization, Inc.
'
' This example formulates and solves the following simple MIP model:
'
'    maximize    x +   y + 2 z
'    subject to  x + 2 y + 3 z <= 4
'                x +   y       >= 1
'    x, y, z binary

Imports System
Imports Gurobi

Class mip1_vb
    Shared Sub Main()
        Try
            Dim env As GRBEnv = New GRBEnv("mip1.log")
            Dim model As GRBModel = New GRBModel(env)

            ' Create variables

            Dim x As GRBVar = model.AddVar(0.0, 1.0, 0.0, GRB.BINARY, "x")
            Dim y As GRBVar = model.AddVar(0.0, 1.0, 0.0, GRB.BINARY, "y")
            Dim z As GRBVar = model.AddVar(0.0, 1.0, 0.0, GRB.BINARY, "z")

            ' Set objective: maximize x + y + 2 z

            model.SetObjective(x + y + 2 * z, GRB.MAXIMIZE)

            ' Add constraint: x + 2 y + 3 z <= 4

            model.AddConstr(x + 2 * y + 3 * z <= 4.0, "c0")

            ' Add constraint: x + y >= 1

            model.AddConstr(x + 2 * y + 3 * z <= 4.0, "c1")

            ' Optimize model

            model.Optimize()

            Console.WriteLine(x.VarName & " " & x.X)
            Console.WriteLine(y.VarName & " " & y.X)
            Console.WriteLine(z.VarName & " " & z.X)
```

```
            Console.WriteLine("Obj: " & model.ObjVal)

            ' Dispose of model and env

            model.Dispose()
            env.Dispose()

        Catch e As GRBException
            Console.WriteLine("Error code: " & e.ErrorCode & ". " & e.Message)
        End Try
    End Sub
End Class
```

mip2_vb.vb

```
' Copyright 2017, Gurobi Optimization, Inc.
'
' This example reads a MIP model from a file, solves it and
' prints the objective values from all feasible solutions
' generated while solving the MIP. Then it creates the fixed
' model and solves that model.


Imports System
Imports Gurobi

Class mip2_vb
    Shared Sub Main(ByVal args As String())

        If args.Length < 1 Then
            Console.WriteLine("Usage: mip2_vb filename")
            Return
        End If

        Try
            Dim env As GRBEnv = New GRBEnv("lp1.log")
            Dim model As GRBModel = New GRBModel(env, args(0))

            If model.IsMIP = 0 Then
                Console.WriteLine("Model is not a MIP")
                Return
            End If

            model.Optimize()

            Dim optimstatus As Integer = model.Status

            If optimstatus = GRB.Status.INF_OR_UNBD Then
                model.Parameters.Presolve = 0
                model.Optimize()
                optimstatus = model.Status
            End If

            Dim objval As Double

            If optimstatus = GRB.Status.OPTIMAL Then
                objval = model.ObjVal
                Console.WriteLine("Optimal objective: " & objval)
            ElseIf optimstatus = GRB.Status.INFEASIBLE Then
```

```
    Console.WriteLine("Model is infeasible")
    model.ComputeIIS()
    model.Write("model.ilp")
    Return
ElseIf optimstatus = GRB.Status.UNBOUNDED Then
    Console.WriteLine("Model is unbounded")
    Return
Else
    Console.WriteLine("Optimization was stopped with status = " & _
                      optimstatus)
    Return
End If

' Iterate over the solutions and compute the objectives
Dim vars() As GRBVar = model.GetVars()
model.Parameters.OutputFlag = 0

Console.WriteLine()
For k As Integer = 0 To model.SolCount - 1
    model.Parameters.SolutionNumber = k
    Dim objn As Double = 0.0

    For j As Integer = 0 To vars.Length - 1
        objn += vars(j).Obj * vars(j).Xn
    Next

    Console.WriteLine("Solution " & k & " has objective: " & objn)
Next
Console.WriteLine()
model.Parameters.OutputFlag = 1

' Solve fixed model
Dim fixedmodel As GRBModel = model.FixedModel()
fixedmodel.Parameters.Presolve = 0
fixedmodel.Optimize()

Dim foptimstatus As Integer = fixedmodel.Status
If foptimstatus <> GRB.Status.OPTIMAL Then
    Console.WriteLine("Error: fixed model isn't optimal")
    Return
End If

Dim fobjval As Double = fixedmodel.ObjVal

If Math.Abs(fobjval - objval) > 0.000001 * (1.0 + Math.Abs(objval)) Then
```

```
            End If

            Dim fvars() As GRBVar = fixedmodel.GetVars()
            Dim x() As Double = fixedmodel.Get(GRB.DoubleAttr.X, fvars)
            Dim vnames() As String = fixedmodel.Get(GRB.StringAttr.VarName, fvars)

            For j As Integer = 0 To fvars.Length - 1
                If x(j) <> 0 Then
                    Console.WriteLine(vnames(j) & " " & x(j))
                End If
            Next

            ' Dispose of models and env
            fixedmodel.Dispose()
            model.Dispose()
            env.Dispose()

        Catch e As GRBException
            Console.WriteLine("Error code: " & e.ErrorCode & ". " & e.Message)
        End Try
    End Sub
End Class
```

multiobj_vb.vb

```
' Copyright 2017, Gurobi Optimization, Inc.

' Want to cover three different sets but subject to a common budget of
' elements allowed to be used. However, the sets have different priorities to
' be covered; and we tackle this by using multi-objective optimization.

Imports Gurobi

Class multiobj_vb

    Shared Sub Main()

        Try
            ' Sample data
            Dim groundSetSize As Integer = 20
            Dim nSubsets As Integer = 4
            Dim Budget As Integer = 12

            Dim [Set] As Double(,) = New Double(,) { _
                {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, _
                {0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1}, _
                {0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0}, _
                {0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0}}

            Dim SetObjPriority As Integer() = New Integer() {3, 2, 2, 1}
            Dim SetObjWeight As Double() = New Double() {1.0, 0.25, 1.25, 1.0}
            Dim e As Integer, i As Integer, status As Integer, nSolutions As Integer

            ' Create environment
            Dim env As New GRBEnv("multiobj_vb.log")

            ' Create initial model
            Dim model As New GRBModel(env)
            model.ModelName = "multiobj_vb"

            ' Initialize decision variables for ground set:
            ' x[e] == 1 if element e is chosen for the covering.
            Dim Elem As GRBVar() = model.AddVars(groundSetSize, GRB.BINARY)
            For e = 0 To groundSetSize - 1
                Dim vname As String = "El" & e.ToString()
                Elem(e).VarName = vname
            Next
```

```
' Constraint: limit total number of elements to be picked to be at most
' Budget
Dim lhs As New GRBLinExpr()
For e = 0 To groundSetSize - 1
    lhs.AddTerm(1.0, Elem(e))
Next
model.AddConstr(lhs, GRB.LESS_EQUAL, Budget, "Budget")

' Set global sense for ALL objectives
model.ModelSense = GRB.MAXIMIZE

' Limit how many solutions to collect
model.Parameters.PoolSolutions = 100

' Set and configure i-th objective
For i = 0 To nSubsets - 1
    Dim vname As String = String.Format("Set{0}", i)
    Dim objn As New GRBLinExpr()
    For e = 0 To groundSetSize - 1
        objn.AddTerm([Set](i, e), Elem(e))
    Next

    model.SetObjectiveN(objn, i, SetObjPriority(i), SetObjWeight(i), _
                        1.0 + i, 0.01, vname)
Next

' Save problem
model.Write("multiobj_vb.lp")

' Optimize
model.Optimize()

' Status checking
status = model.Status

If status = GRB.Status.INF_OR_UNBD OrElse _
   status = GRB.Status.INFEASIBLE OrElse _
   status = GRB.Status.UNBOUNDED Then
    Console.WriteLine("The model cannot be solved " & _
             "because it is infeasible or unbounded")
    Return
End If
If status <> GRB.Status.OPTIMAL Then
    Console.WriteLine("Optimization was stopped with status {0}", status)
    Return
```

```
            End If

            ' Print best selected set
            Console.WriteLine("Selected elements in best solution:")
            Console.Write(vbTab)
            For e = 0 To groundSetSize - 1
                If Elem(e).X < 0.9 Then
                    Continue For
                End If
                Console.Write("El{0} ", e)
            Next
            Console.WriteLine()

            ' Print number of solutions stored
            nSolutions = model.SolCount
            Console.WriteLine("Number of solutions found: {0}", nSolutions)

            ' Print objective values of solutions
            If nSolutions > 10 Then
                nSolutions = 10
            End If
            Console.WriteLine("Objective values for first {0} solutions:", nSolutions)
            For i = 0 To nSubsets - 1
                model.Parameters.ObjNumber = i

                Console.Write(vbTab & "Set" & i)
                For e = 0 To nSolutions - 1
                    model.Parameters.SolutionNumber = e
                    Console.Write("{0,8}", model.ObjNVal)
                Next
                Console.WriteLine()
            Next

            model.Dispose()
            env.Dispose()

        Catch e As GRBException
            Console.WriteLine("Error code = {0}", e)
            Console.WriteLine(e.Message)

        End Try
    End Sub
End Class
```

params_vb.vb

```
' Copyright 2017, Gurobi Optimization, Inc. */

' Use parameters that are associated with a model.

' A MIP is solved for a few seconds with different sets of parameters.
' The one with the smallest MIP gap is selected, and the optimization
' is resumed until the optimal solution is found.

Imports System
Imports Gurobi

Class params_vb
    Shared Sub Main(args As String())
        If args.Length < 1 Then
            Console.Out.WriteLine("Usage: params_vb filename")
            Return
        End If

        Try
            ' Read model and verify that it is a MIP
            Dim env As New GRBEnv()
            Dim m As New GRBModel(env, args(0))
            If m.IsMIP = 0 Then
                Console.WriteLine("The model is not an integer program")
                Environment.Exit(1)
            End If

            ' Set a 2 second time limit
            m.Parameters.TimeLimit = 2.0

            ' Now solve the model with different values of MIPFocus
            Dim bestModel As New GRBModel(m)
            bestModel.Optimize()
            For i As Integer = 1 To 3
                m.Reset()
                m.Parameters.MIPFocus = i
                m.Optimize()
                If bestModel.MIPGap > m.MIPGap Then
                    Dim swap As GRBModel = bestModel
                    bestModel = m
                    m = swap
                End If
            Next
```

```
            ' Finally, delete the extra model, reset the time limit and
            ' continue to solve the best model to optimality
            m.Dispose()
            bestModel.Parameters.TimeLimit = GRB.INFINITY
            bestModel.Optimize()

            Console.WriteLine("Solved with MIPFocus: " & bestModel.Parameters.MIPFocus)
        Catch e As GRBException
            Console.WriteLine("Error code: " + e.ErrorCode & ". " + e.Message)
        End Try
    End Sub
End Class
```

piecewise_vb.vb

```
' Copyright 2017, Gurobi Optimization, Inc.

' This example considers the following separable, convex problem:
'
'    minimize    f(x) - y + g(z)
'    subject to  x + 2 y + 3 z <= 4
'                x +   y       >= 1
'                x,    y,    z <= 1
'
' where f(u) = exp(-u) and g(u) = 2 u^2 - 4 u, for all real u. It
' formulates and solves a simpler LP model by approximating f and
' g with piecewise-linear functions. Then it transforms the model
' into a MIP by negating the approximation for f, which corresponds
' to a non-convex piecewise-linear function, and solves it again.

Imports System
Imports Gurobi

Class piecewise_vb
    Shared Function f(u As Double) As Double
        Return Math.Exp(-u)
    End Function
    Shared Function g(u As Double) As Double
        Return 2 * u * u - 4 * u
    End Function

    Shared Sub Main()
        Try
            ' Create environment

            Dim env As New GRBEnv()

            ' Create a new model

            Dim model As New GRBModel(env)

            ' Create variables

            Dim lb As Double = 0.0, ub As Double = 1.0

            Dim x As GRBVar = model.AddVar(lb, ub, 0.0, GRB.CONTINUOUS, "x")
            Dim y As GRBVar = model.AddVar(lb, ub, 0.0, GRB.CONTINUOUS, "y")
            Dim z As GRBVar = model.AddVar(lb, ub, 0.0, GRB.CONTINUOUS, "z")
```

```
' Set objective for y

model.SetObjective(-y)

' Add piecewise-linear objective functions for x and z

Dim npts As Integer = 101
Dim ptu As Double() = New Double(npts - 1) {}
Dim ptf As Double() = New Double(npts - 1) {}
Dim ptg As Double() = New Double(npts - 1) {}

For i As Integer = 0 To npts - 1
    ptu(i) = lb + (ub - lb) * i / (npts - 1)
    ptf(i) = f(ptu(i))
    ptg(i) = g(ptu(i))
Next

model.SetPWLObj(x, ptu, ptf)
model.SetPWLObj(z, ptu, ptg)

' Add constraint: x + 2 y + 3 z <= 4

model.AddConstr(x + 2 * y + 3 * z <= 4.0, "c0")

' Add constraint: x + y >= 1

model.AddConstr(x + y >= 1.0, "c1")

' Optimize model as an LP

model.Optimize()

Console.WriteLine("IsMIP: " & model.IsMIP)

Console.WriteLine(x.VarName & " " & x.X)
Console.WriteLine(y.VarName & " " & y.X)
Console.WriteLine(z.VarName & " " & z.X)

Console.WriteLine("Obj: " & model.ObjVal)

Console.WriteLine()

' Negate piecewise-linear objective function for x

For i As Integer = 0 To npts - 1
```

```
                ptf(i) = -ptf(i)
            Next

            model.SetPWLObj(x, ptu, ptf)

            ' Optimize model as a MIP

            model.Optimize()

            Console.WriteLine("IsMIP: " & model.IsMIP)

            Console.WriteLine(x.VarName & " " & x.X)
            Console.WriteLine(y.VarName & " " & y.X)
            Console.WriteLine(z.VarName & " " & z.X)

            Console.WriteLine("Obj: " & model.ObjVal)

            ' Dispose of model and environment

            model.Dispose()

            env.Dispose()
        Catch e As GRBException
            Console.WriteLine("Error code: " + e.ErrorCode & ". " + e.Message)
        End Try
    End Sub
End Class
```

poolsearch_vb.vb

```
' Copyright 2017, Gurobi Optimization, Inc.

' We find alternative epsilon-optimal solutions to a given knapsack
' problem by using PoolSearchMode

Imports Gurobi

Class poolsearch_vb

    Shared Sub Main()

        Try

            'Sample data
            Dim groundSetSize As Integer = 10

            Dim objCoef As Double() = New Double() { _
                32, 32, 15, 15, 6, 6, 1, 1, 1, 1}

            Dim knapsackCoef As Double() = New Double() { _
                16, 16, 8, 8, 4, 4, 2, 2, 1, 1}

            Dim Budget As Double = 33
            Dim e As Integer, status As Integer, nSolutions As Integer

            ' Create environment
            Dim env As New GRBEnv("poolsearch_vb.log")

            ' Create initial model
            Dim model As New GRBModel(env)
            model.ModelName = "poolsearch_vb"

            ' Initialize decision variables for ground set:
            ' x[e] == k if element e is chosen k-times.
            Dim Elem As GRBVar() = model.AddVars(groundSetSize, GRB.BINARY)
            model.[Set](GRB.DoubleAttr.Obj, Elem, objCoef, 0, groundSetSize)

            For e = 0 To groundSetSize - 1
                Elem(e).VarName = String.Format("El{0}", e)
            Next

            ' Constraint: limit total number of elements to be picked to be at most Budget
            Dim lhs As New GRBLinExpr()
```

```
For e = 0 To groundSetSize - 1
    lhs.AddTerm(knapsackCoef(e), Elem(e))
Next
model.AddConstr(lhs, GRB.LESS_EQUAL, Budget, "Budget")

' set global sense for ALL objectives
model.ModelSense = GRB.MAXIMIZE

' Limit how many solutions to collect
model.Parameters.PoolSolutions = 1024

' Limit how many solutions to collect
model.Parameters.PoolGap = 0.1

' Limit how many solutions to collect
model.Parameters.PoolSearchMode = 2

' save problem
model.Write("poolsearch_vb.lp")

' Optimize
model.Optimize()

' Status checking
status = model.Status

If status = GRB.Status.INF_OR_UNBD OrElse _
   status = GRB.Status.INFEASIBLE OrElse _
   status = GRB.Status.UNBOUNDED Then
    Console.WriteLine("The model cannot be solved because it is infeasible or unboun
    Return
End If
If status <> GRB.Status.OPTIMAL Then
    Console.WriteLine("Optimization was stopped with status {0}", status)
    Return
End If

' Print best selected set
Console.WriteLine("Selected elements in best solution:")
Console.Write(vbTab)
For e = 0 To groundSetSize - 1
    If Elem(e).X < 0.9 Then
        Continue For
    End If
    Console.Write("El{0} ", e)
```

```
            Next
            Console.WriteLine()

            ' Print number of solutions stored
            nSolutions = model.SolCount
            Console.WriteLine("Number of solutions found: ", nSolutions)

            ' Print objective values of solutions
            For e = 0 To nSolutions - 1
                model.Parameters.SolutionNumber = e
                Console.Write("{0} ", model.PoolObjVal)
                If e Mod 15 = 14 Then
                    Console.WriteLine()
                End If
            Next
            Console.WriteLine()

            model.Dispose()
            env.Dispose()

        Catch e As GRBException
            Console.WriteLine("Error code: {0}. {1}", e.ErrorCode, e.Message)

        End Try
    End Sub

End Class
```

qcp_vb.vb

```
' Copyright 2017, Gurobi Optimization, Inc.

' This example formulates and solves the following simple QCP model:
'
'     maximize    x
'     subject to  x + y + z = 1
'                 x^2 + y^2 <= z^2 (second-order cone)
'                 x^2 <= yz        (rotated second-order cone)

Imports Gurobi

Class qcp_vb
    Shared Sub Main()
        Try
            Dim env As New GRBEnv("qcp.log")
            Dim model As New GRBModel(env)

            ' Create variables

            Dim x As GRBVar = model.AddVar(0.0, GRB.INFINITY, 0.0, GRB.CONTINUOUS, "x")
            Dim y As GRBVar = model.AddVar(0.0, GRB.INFINITY, 0.0, GRB.CONTINUOUS, "y")
            Dim z As GRBVar = model.AddVar(0.0, GRB.INFINITY, 0.0, GRB.CONTINUOUS, "z")

            ' Set objective

            Dim obj As GRBLinExpr = x
            model.SetObjective(obj, GRB.MAXIMIZE)

            ' Add linear constraint: x + y + z = 1

            model.AddConstr(x + y + z = 1.0, "c0")

            ' Add second-order cone: x^2 + y^2 <= z^2

            model.AddQConstr(x * x + y * y <= z * z, "qc0")

            ' Add rotated cone: x^2 <= yz

            model.AddQConstr(x * x <= y * z, "qc1")

            ' Optimize model

            model.Optimize()
```

```
            Console.WriteLine(x.VarName & " " & x.X)
            Console.WriteLine(y.VarName & " " & y.X)
            Console.WriteLine(z.VarName & " " & z.X)

            Console.WriteLine("Obj: " & model.ObjVal & " " & obj.Value)

            ' Dispose of model and env

            model.Dispose()

            env.Dispose()
        Catch e As GRBException
            Console.WriteLine("Error code: " & e.ErrorCode & ". " & e.Message)
        End Try
    End Sub
End Class
```

qp_vb.vb

```
' Copyright 2017, Gurobi Optimization, Inc.

' This example formulates and solves the following simple QP model:
'
'     minimize    x^2 + x*y + y^2 + y*z + z^2 + 2 x
'     subject to  x + 2 y + 3 z >= 4
'                 x +   y       >= 1
'
'   It solves it once as a continuous model, and once as an integer model.
'

Imports Gurobi

Class qp_vb
    Shared Sub Main()
        Try
            Dim env As New GRBEnv("qp.log")
            Dim model As New GRBModel(env)

            ' Create variables

            Dim x As GRBVar = model.AddVar(0.0, 1.0, 0.0, GRB.CONTINUOUS, "x")
            Dim y As GRBVar = model.AddVar(0.0, 1.0, 0.0, GRB.CONTINUOUS, "y")
            Dim z As GRBVar = model.AddVar(0.0, 1.0, 0.0, GRB.CONTINUOUS, "z")

            ' Set objective

            Dim obj As New GRBQuadExpr()
            obj = x*x + x*y + y*y + y*z + z*z + 2*x
            model.SetObjective(obj)

            ' Add constraint: x + 2 y + 3 z >= 4

            model.AddConstr(x + 2 * y + 3 * z >= 4.0, "c0")

            ' Add constraint: x + y >= 1

            model.AddConstr(x + y >= 1.0, "c1")

            ' Optimize model

            model.Optimize()

            Console.WriteLine(x.VarName & " " & x.X)
```

```
            Console.WriteLine(y.VarName & " " & y.X)
            Console.WriteLine(z.VarName & " " & z.X)

            Console.WriteLine("Obj: " & model.ObjVal & " " & obj.Value)


            ' Change variable types to integer

            x.VType = GRB.INTEGER
            y.VType = GRB.INTEGER
            z.VType = GRB.INTEGER

            ' Optimize model

            model.Optimize()

            Console.WriteLine(x.VarName & " " & x.X)
            Console.WriteLine(y.VarName & " " & y.X)
            Console.WriteLine(z.VarName & " " & z.X)

            Console.WriteLine("Obj: " & model.ObjVal & " " & obj.Value)

            ' Dispose of model and env

            model.Dispose()
            env.Dispose()

        Catch e As GRBException
            Console.WriteLine("Error code: " & e.ErrorCode & ". " & e.Message)
        End Try
    End Sub
End Class
```

sensitivity_vb.vb

```
' Copyright 2017, Gurobi Optimization, Inc.

' A simple sensitivity analysis example which reads a MIP model
' from a file and solves it. Then each binary variable is set
' to 1-X, where X is its value in the optimal solution, and
' the impact on the objective function value is reported.

Imports System
Imports Gurobi

Class sensitivity_vb
    Shared Sub Main(args As String())
        If args.Length < 1 Then
            Console.Out.WriteLine("Usage: sensitivity_vb filename")
            Return
        End If

        Try

            ' Create environment

            Dim env As New GRBEnv()

            ' Read and solve model

            Dim model As New GRBModel(env, args(0))

            If model.IsMIP = 0 Then
                Console.WriteLine("Model is not a MIP")
                Return
            End If

            model.Optimize()

            If model.Status <> GRB.Status.OPTIMAL Then
                Console.WriteLine("Optimization ended with status " & _
                                  model.Status)
                Return
            End If

            ' Store the optimal solution

            Dim origObjVal As Double = model.ObjVal
            Dim vars As GRBVar() = model.GetVars()
```

```
Dim origX As Double() = model.Get(GRB.DoubleAttr.X, vars)

' Disable solver output for subsequent solves

model.Parameters.OutputFlag = 0

' Iterate through unfixed, binary variables in model

For i As Integer = 0 To vars.Length - 1
    Dim v As GRBVar = vars(i)
    Dim vType As Char = v.VType

    If v.LB = 0 AndAlso _
       v.UB = 1 AndAlso _
       (vType = GRB.BINARY OrElse vType = GRB.INTEGER) Then

        ' Set variable to 1-X, where X is its value in optimal solution

        If origX(i) < 0.5 Then
            v.LB = 1.0
            v.Start = 1.0
        Else
            v.UB = 0.0
            v.Start = 0.0
        End If

        ' Update MIP start for the other variables

        For j As Integer = 0 To vars.Length - 1
            If j <> i Then
                vars(j).Start = origX(j)
            End If
        Next

        ' Solve for new value and capture sensitivity information

        model.Optimize()

        If model.Status = GRB.Status.OPTIMAL Then
            Console.WriteLine("Objective sensitivity for variable " & _
                              v.VarName & " is " & (model.ObjVal - origObjVal))
        Else
            Console.WriteLine("Objective sensitivity for variable " & _
                              v.VarName & " is infinite")
        End If
```

```
                    ' Restore the original variable bounds

                    v.LB = 0.0
                    v.UB = 1.0
                End If
            Next

            ' Dispose of model and environment

            model.Dispose()

            env.Dispose()
        Catch e As GRBException
            Console.WriteLine("Error code: " + e.ErrorCode)
            Console.WriteLine(e.Message)
            Console.WriteLine(e.StackTrace)
        End Try
    End Sub
End Class
```

sos_vb.vb

```
' Copyright 2017, Gurobi Optimization, Inc.
'
' This example creates a very simple Special Ordered Set (SOS) model.
' The model consists of 3 continuous variables, no linear constraints,
' and a pair of SOS constraints of type 1.

Imports System
Imports Gurobi

Class sos_vb
    Shared Sub Main()
        Try
            Dim env As New GRBEnv()
            Dim model As New GRBModel(env)

            ' Create variables

            Dim ub As Double() = {1, 1, 2}
            Dim obj As Double() = {-2, -1, -1}
            Dim names As String() = {"x0", "x1", "x2"}

            Dim x As GRBVar() = model.AddVars(Nothing, ub, obj, Nothing, names)

            ' Add first SOS1: x0=0 or x1=0

            Dim sosv1 As GRBVar() = {x(0), x(1)}
            Dim soswt1 As Double() = {1, 2}

            model.AddSOS(sosv1, soswt1, GRB.SOS_TYPE1)

            ' Add second SOS1: x0=0 or x2=0

            Dim sosv2 As GRBVar() = {x(0), x(2)}
            Dim soswt2 As Double() = {1, 2}

            model.AddSOS(sosv2, soswt2, GRB.SOS_TYPE1)

            ' Optimize model

            model.Optimize()

            For i As Integer = 0 To 2
                Console.WriteLine(x(i).VarName & " " & x(i).X)
            Next
```

```
            ' Dispose of model and env
            model.Dispose()
            env.Dispose()

        Catch e As GRBException
            Console.WriteLine("Error code: " & e.ErrorCode & ". " & e.Message)
        End Try
    End Sub
End Class
```

sudoku_vb.vb

```
' Copyright 2017, Gurobi Optimization, Inc.
'
' Sudoku example.
'
' The Sudoku board is a 9x9 grid, which is further divided into a 3x3 grid
' of 3x3 grids.  Each cell in the grid must take a value from 0 to 9.
' No two grid cells in the same row, column, or 3x3 subgrid may take the
' same value.

' In the MIP formulation, binary variables x(i,j,v) indicate whether
' cell <i,j> takes value 'v'.  The constraints are as follows:
'   1. Each cell must take exactly one value (sum_v x(i,j,v) = 1)
'   2. Each value is used exactly once per row (sum_i x(i,j,v) = 1)
'   3. Each value is used exactly once per column (sum_j x(i,j,v) = 1)
'   4. Each value is used exactly once per 3x3 subgrid (sum_grid x(i,j,v) = 1)
'
' Input datasets for this example can be found in examples/data/sudoku*.

Imports System
Imports System.IO
Imports Gurobi

Class sudoku_vb
    Shared Sub Main(ByVal args as String())
        Dim n As Integer = 9
        Dim s As Integer = 3

        If args.Length < 1 Then
            Console.WriteLine("Usage: sudoku_vb filename")
            Return
        End If

        Try
            Dim env As New GRBEnv()
            Dim model As New GRBModel(env)

            ' Create 3-D array of model variables

            Dim vars As GRBVar(,,) = New GRBVar(n - 1, n - 1, n - 1) {}

            For i As Integer = 0 To n - 1
                For j As Integer = 0 To n - 1
                    For v As Integer = 0 To n - 1
                        Dim st As String = "G_" & i & "_" & j & "_" & v
```

```
            vars(i, j, v) = model.AddVar(0.0, 1.0, 0.0, GRB.BINARY, st)
        Next
    Next
Next

' Add constraints

Dim expr As GRBLinExpr

' Each cell must take one value

For i As Integer = 0 To n - 1
    For j As Integer = 0 To n - 1
        expr = 0
        For v As Integer = 0 To n - 1
            expr.AddTerm(1.0, vars(i, j, v))
        Next
        Dim st As String = "V_" & i & "_" & j
        model.AddConstr(expr = 1, st)
    Next
Next

' Each value appears once per row

For i As Integer = 0 To n - 1
    For v As Integer = 0 To n - 1
        expr = 0
        For j As Integer = 0 To n - 1
            expr.AddTerm(1.0, vars(i, j, v))
        Next
        Dim st As String = "R_" & i & "_" & v
        model.AddConstr(expr = 1, st)
    Next
Next

' Each value appears once per column

For j As Integer = 0 To n - 1
    For v As Integer = 0 To n - 1
        expr = 0
        For i As Integer = 0 To n - 1
            expr.AddTerm(1.0, vars(i, j, v))
        Next
        Dim st As String = "C_" & j & "_" & v
        model.AddConstr(expr = 1, st)
```

```
    Next
Next

' Each value appears once per sub-grid

For v As Integer = 0 To n - 1
    For i0 As Integer = 0 To s - 1
        For j0 As Integer = 0 To s - 1
            expr = 0
            For i1 As Integer = 0 To s - 1
                For j1 As Integer = 0 To s - 1
                    expr.AddTerm(1.0, vars(i0 * s + i1, j0 * s + j1, v))
                Next
            Next
            Dim st As String = "Sub_" & v & "_" & i0 & "_" & j0
            model.AddConstr(expr = 1, st)
        Next
    Next
Next

' Fix variables associated with pre-specified cells

Dim sr As StreamReader = File.OpenText(args(0))

For i As Integer = 0 To n - 1
    Dim input As String = sr.ReadLine()
    For j As Integer = 0 To n - 1
        Dim val As Integer = Microsoft.VisualBasic.Asc(input(j)) - 48 - 1
        ' 0-based
        If val >= 0 Then
            vars(i, j, val).LB = 1.0
        End If
    Next
Next

' Optimize model

model.Optimize()

' Write model to file
model.Write("sudoku.lp")

Dim x As Double(,,) = model.Get(GRB.DoubleAttr.X, vars)

Console.WriteLine()
```

```
            For i As Integer = 0 To n - 1
                For j As Integer = 0 To n - 1
                    For v As Integer = 0 To n - 1
                        If x(i, j, v) > 0.5 Then
                            Console.Write(v + 1)
                        End If
                    Next
                Next
                Console.WriteLine()
            Next

            ' Dispose of model and env
            model.Dispose()
            env.Dispose()

        Catch e As GRBException
            Console.WriteLine("Error code: " & e.ErrorCode & ". " & e.Message)
        End Try
    End Sub
End Class
```

tsp_vb.vb

```
' Copyright 2017, Gurobi Optimization, Inc.
'
' Solve a traveling salesman problem on a randomly generated set of
' points using lazy constraints.   The base MIP model only includes
' 'degree-2' constraints, requiring each node to have exactly
' two incident edges.  Solutions to this model may contain subtours -
' tours that don't visit every node.  The lazy constraint callback
' adds new constraints to cut them off.

Imports Gurobi

Class tsp_vb
    Inherits GRBCallback
    Private vars As GRBVar(,)

    Public Sub New(xvars As GRBVar(,))
        vars = xvars
    End Sub

    ' Subtour elimination callback.    Whenever a feasible solution is found,
    ' find the smallest subtour, and add a subtour elimination constraint
    ' if the tour doesn't visit every node.

    Protected Overrides Sub Callback()
        Try
            If where = GRB.Callback.MIPSOL Then
                ' Found an integer feasible solution - does it visit every node?

                Dim n As Integer = vars.GetLength(0)
                Dim tour As Integer() = findsubtour(GetSolution(vars))

                If tour.Length < n Then
                    ' Add subtour elimination constraint
                    Dim expr As GRBLinExpr = 0
                    For i As Integer = 0 To tour.Length - 1
                        For j As Integer = i + 1 To tour.Length - 1
                            expr.AddTerm(1.0, vars(tour(i), tour(j)))
                        Next
                    Next
                    AddLazy(expr <= tour.Length - 1)
                End If
            End If
        Catch e As GRBException
            Console.WriteLine("Error code: " & e.ErrorCode & ". " & e.Message)
```

```
        Console.WriteLine(e.StackTrace)
    End Try
End Sub

' Given an integer-feasible solution 'sol', returns the smallest
' sub-tour (as a list of node indices).

Protected Shared Function findsubtour(sol As Double(,)) As Integer()
    Dim n As Integer = sol.GetLength(0)
    Dim seen As Boolean() = New Boolean(n - 1) {}
    Dim tour As Integer() = New Integer(n - 1) {}
    Dim bestind As Integer, bestlen As Integer
    Dim i As Integer, node As Integer, len As Integer, start As Integer

    For i = 0 To n - 1
        seen(i) = False
    Next

    start = 0
    bestlen = n+1
    bestind = -1
    node = 0
    While start < n
        For node = 0 To n - 1
            if Not seen(node)
                Exit For
            End if
        Next
        if node = n
            Exit While
        End if
        For len = 0 To n - 1
            tour(start+len) = node
            seen(node) = true
            For i = 0 To n - 1
                if sol(node, i) > 0.5 AndAlso Not seen(i)
                    node = i
                    Exit For
                End If
            Next
            If i = n
                len = len + 1
                If len < bestlen
                    bestlen = len
                    bestind = start
```

```
                End If
                start = start + len
                Exit For
            End If
        Next
    End While

    For i = 0 To bestlen - 1
      tour(i) = tour(bestind+i)
    Next
    System.Array.Resize(tour, bestlen)

    Return tour
End Function

' Euclidean distance between points 'i' and 'j'

Protected Shared Function distance(x As Double(), y As Double(), _
                                   i As Integer, j As Integer) As Double
    Dim dx As Double = x(i) - x(j)
    Dim dy As Double = y(i) - y(j)
    Return Math.Sqrt(dx * dx + dy * dy)
End Function

Public Shared Sub Main(args As String())

    If args.Length < 1 Then
        Console.WriteLine("Usage: tsp_vb nnodes")
        Return
    End If

    Dim n As Integer = Convert.ToInt32(args(0))

    Try
        Dim env As New GRBEnv()
        Dim model As New GRBModel(env)

        ' Must set LazyConstraints parameter when using lazy constraints

        model.Parameters.LazyConstraints = 1

        Dim x As Double() = New Double(n - 1) {}
        Dim y As Double() = New Double(n - 1) {}

        Dim r As New Random()
```

```
For i As Integer = 0 To n - 1
    x(i) = r.NextDouble()
    y(i) = r.NextDouble()
Next

' Create variables

Dim vars As GRBVar(,) = New GRBVar(n - 1, n - 1) {}

For i As Integer = 0 To n - 1
    For j As Integer = 0 To i
        vars(i, j) = model.AddVar(0.0, 1.0, distance(x, y, i, j), _
                                  GRB.BINARY, "x" & i & "_" & j)
        vars(j, i) = vars(i, j)
    Next
Next

' Degree-2 constraints

For i As Integer = 0 To n - 1
    Dim expr As GRBLinExpr = 0
    For j As Integer = 0 To n - 1
        expr.AddTerm(1.0, vars(i, j))
    Next
    model.AddConstr(expr = 2.0, "deg2_" & i)
Next

' Forbid edge from node back to itself

For i As Integer = 0 To n - 1
    vars(i, i).UB = 0.0
Next

model.SetCallback(New tsp_vb(vars))
model.Optimize()

If model.SolCount > 0 Then
    Dim tour As Integer() = findsubtour(model.Get(GRB.DoubleAttr.X, vars))

    Console.Write("Tour: ")
    For i As Integer = 0 To tour.Length - 1
        Console.Write(tour(i) & " ")
    Next
    Console.WriteLine()
End If
```

```
            ' Dispose of model and environment
            model.Dispose()

            env.Dispose()
        Catch e As GRBException
            Console.WriteLine("Error code: " & e.ErrorCode & ". " & e.Message)
            Console.WriteLine(e.StackTrace)
        End Try
    End Sub
End Class
```

tune_vb.vb

```
' Copyright 2017, Gurobi Optimization, Inc. */
'
' This example reads a model from a file and tunes it.
' It then writes the best parameter settings to a file
' and solves the model using these parameters.

Imports System
Imports Gurobi

Class tune_vb
    Shared Sub Main(ByVal args As String())

        If args.Length < 1 Then
            Console.Out.WriteLine("Usage: tune_vb filename")
            Return
        End If

        Try
            Dim env As New GRBEnv()

            ' Read model from file
            Dim model As New GRBModel(env, args(0))

            ' Set the TuneResults parameter to 1
            model.Parameters.TuneResults = 1

            ' Tune the model
            model.Tune()

            ' Get the number of tuning results
            Dim resultcount As Integer = model.TuneResultCount

            If resultcount > 0 Then

                ' Load the tuned parameters into the model's environment
                model.GetTuneResult(0)

                ' Write the tuned parameters to a file
                model.Write("tune.prm")

                ' Solve the model using the tuned parameters
                model.Optimize()
            End If
```

```
            ' Dispose of model and environment
            model.Dispose()
            env.Dispose()

        Catch e As GRBException
            Console.WriteLine("Error code: " & e.ErrorCode & ". " & e.Message)
        End Try
    End Sub
End Class
```

workforce1_vb.vb

```
' Copyright 2017, Gurobi Optimization, Inc.
'
' Assign workers to shifts; each worker may or may not be available on a
' particular day. If the problem cannot be solved, use IIS to find a set of
' conflicting constraints. Note that there may be additional conflicts
' besides what is reported via IIS.

Imports System
Imports Gurobi

Class workforce1_vb
    Shared Sub Main()
        Try

            ' Sample data
            ' Sets of days and workers
            Dim Shifts As String() = New String() {"Mon1", "Tue2", "Wed3", "Thu4", _
                                                   "Fri5", "Sat6", "Sun7", "Mon8", _
                                                   "Tue9", "Wed10", "Thu11", _
                                                   "Fri12", "Sat13", "Sun14"}
            Dim Workers As String() = New String() {"Amy", "Bob", "Cathy", "Dan", _
                                                    "Ed", "Fred", "Gu"}

            Dim nShifts As Integer = Shifts.Length
            Dim nWorkers As Integer = Workers.Length

            ' Number of workers required for each shift
            Dim shiftRequirements As Double() = New Double() {3, 2, 4, 4, 5, 6, _
                                                              5, 2, 2, 3, 4, 6, _
                                                              7, 5}

            ' Amount each worker is paid to work one shift
            Dim pay As Double() = New Double() {10, 12, 10, 8, 8, 9, 11}

            ' Worker availability: 0 if the worker is unavailable for a shift
            Dim availability As Double(,) = New Double(,) { _
                      {0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1}, _
                      {1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0}, _
                      {0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1}, _
                      {0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1}, _
                      {1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1}, _
                      {1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1}, _
                      {1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}}
```

```
' Model
Dim env As New GRBEnv()
Dim model As New GRBModel(env)

model.ModelName = "assignment"

' Assignment variables: x(w)(s) == 1 if worker w is assigned
' to shift s. Since an assignment model always produces integer
' solutions, we use continuous variables and solve as an LP.
Dim x As GRBVar(,) = New GRBVar(nWorkers - 1, nShifts - 1) {}
For w As Integer = 0 To nWorkers - 1
    For s As Integer = 0 To nShifts - 1
        x(w, s) = model.AddVar(0, availability(w, s), pay(w), _
                               GRB.CONTINUOUS, _
                               Workers(w) & "." & Shifts(s))
    Next
Next

' The objective is to minimize the total pay costs
model.ModelSense = GRB.MINIMIZE

' Constraint: assign exactly shiftRequirements(s) workers
' to each shift s
For s As Integer = 0 To nShifts - 1
    Dim lhs As GRBLinExpr = 0
    For w As Integer = 0 To nWorkers - 1
        lhs.AddTerm(1.0, x(w, s))
    Next
    model.AddConstr(lhs = shiftRequirements(s), Shifts(s))
Next

' Optimize
model.Optimize()
Dim status As Integer = model.Status
If status = GRB.Status.UNBOUNDED Then
    Console.WriteLine("The model cannot be solved " & _
                      "because it is unbounded")
    Exit Sub
End If
If status = GRB.Status.OPTIMAL Then
    Console.WriteLine("The optimal objective is " & model.ObjVal)
    Exit Sub
End If
If (status <> GRB.Status.INF_OR_UNBD) AndAlso _
   (status <> GRB.Status.INFEASIBLE) Then
```

```
            Console.WriteLine("Optimization was stopped with status " & status)
            Exit Sub
        End If

        ' Do IIS
        Console.WriteLine("The model is infeasible; computing IIS")
        model.ComputeIIS()
        Console.WriteLine(vbLf & "The following constraint(s) " & _
                          "cannot be satisfied:")
        For Each c As GRBConstr In model.GetConstrs()
            If c.IISConstr = 1 Then
                Console.WriteLine(c.ConstrName)
            End If

        Next

        ' Dispose of model and env
        model.Dispose()
        env.Dispose()

    Catch e As GRBException
        Console.WriteLine("Error code: " & e.ErrorCode & ". " & e.Message)
    End Try
  End Sub
End Class
```

workforce2_vb.vb

```
' Copyright 2017, Gurobi Optimization, Inc.
'
' Assign workers to shifts; each worker may or may not be available on a
' particular day. If the problem cannot be solved, use IIS iteratively to
' find all conflicting constraints.

Imports System
Imports System.Collections.Generic
Imports Gurobi

Class workforce2_vb
    Shared Sub Main()
        Try

            ' Sample data
            ' Sets of days and workers
            Dim Shifts As String() = New String() {"Mon1", "Tue2", "Wed3", "Thu4", _
                                                   "Fri5", "Sat6", "Sun7", "Mon8", _
                                                   "Tue9", "Wed10", "Thu11", _
                                                   "Fri12", "Sat13", "Sun14"}
            Dim Workers As String() = New String() {"Amy", "Bob", "Cathy", "Dan", _
                                                    "Ed", "Fred", "Gu"}

            Dim nShifts As Integer = Shifts.Length
            Dim nWorkers As Integer = Workers.Length

            ' Number of workers required for each shift
            Dim shiftRequirements As Double() = New Double() {3, 2, 4, 4, 5, 6, _
                                                              5, 2, 2, 3, 4, 6, _
                                                              7, 5}

            ' Amount each worker is paid to work one shift
            Dim pay As Double() = New Double() {10, 12, 10, 8, 8, 9, 11}

            ' Worker availability: 0 if the worker is unavailable for a shift
            Dim availability As Double(,) = New Double(,) { _
                      {0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1}, _
                      {1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0}, _
                      {0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1}, _
                      {0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1}, _
                      {1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1}, _
                      {1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1}, _
                      {1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}}
```

```
' Model
Dim env As New GRBEnv()
Dim model As New GRBModel(env)

model.ModelName = "assignment"

' Assignment variables: x(w)(s) == 1 if worker w is assigned
' to shift s. Since an assignment model always produces integer
' solutions, we use continuous variables and solve as an LP.
Dim x As GRBVar(,) = New GRBVar(nWorkers - 1, nShifts - 1) {}
For w As Integer = 0 To nWorkers - 1
    For s As Integer = 0 To nShifts - 1
        x(w, s) = model.AddVar(0, availability(w, s), pay(w), _
                               GRB.CONTINUOUS, _
                               Workers(w) & "." & Shifts(s))
    Next
Next

' The objective is to minimize the total pay costs
model.ModelSense = GRB.MINIMIZE

' Constraint: assign exactly shiftRequirements(s) workers
' to each shift s
For s As Integer = 0 To nShifts - 1
    Dim lhs As GRBLinExpr = 0
    For w As Integer = 0 To nWorkers - 1
        lhs.AddTerm(1.0, x(w, s))
    Next
    model.AddConstr(lhs = shiftRequirements(s), Shifts(s))
Next

' Optimize
model.Optimize()
Dim status As Integer = model.Status
If status = GRB.Status.UNBOUNDED Then
    Console.WriteLine("The model cannot be solved " & _
                      "because it is unbounded")
    Exit Sub
End If
If status = GRB.Status.OPTIMAL Then
    Console.WriteLine("The optimal objective is " & model.ObjVal)
    Exit Sub
End If
If (status <> GRB.Status.INF_OR_UNBD) AndAlso _
   (status <> GRB.Status.INFEASIBLE) Then
```

```
    Console.WriteLine("Optimization was stopped with status " & status)
    Exit Sub
End If

' Do IIS
Console.WriteLine("The model is infeasible; computing IIS")
Dim removed As LinkedList(Of String) = New LinkedList(Of String)()

' Loop until we reduce to a model that can be solved
While True
    model.ComputeIIS()
    Console.WriteLine(vbLf & "The following constraint cannot be satisfied:")
    For Each c As GRBConstr In model.GetConstrs()
        If c.IISConstr = 1 Then
            Console.WriteLine(c.ConstrName)
            ' Remove a single constraint from the model
            removed.AddFirst(c.ConstrName)
            model.Remove(c)
            Exit For
        End If
    Next

    Console.WriteLine()
    model.Optimize()
    status = model.Status

    If status = GRB.Status.UNBOUNDED Then
        Console.WriteLine("The model cannot be solved " & _
                          "because it is unbounded")
        Exit Sub
    End If
    If status = GRB.Status.OPTIMAL Then
        Exit While
    End If
    If (status <> GRB.Status.INF_OR_UNBD) AndAlso _
       (status <> GRB.Status.INFEASIBLE) Then
        Console.WriteLine("Optimization was stopped with status " & _
                          status)
        Exit Sub
    End If
End While

Console.WriteLine(vbLf & "The following constraints were removed " & _
                  "to get a feasible LP:")
For Each s As String In removed
```

```
                Console.Write(s & " ")
            Next

            Console.WriteLine()

            ' Dispose of model and env
            model.Dispose()
            env.Dispose()

        Catch e As GRBException
            Console.WriteLine("Error code: " & e.ErrorCode & ". " & e.Message)
        End Try
    End Sub
End Class
```

workforce3_vb.vb

```
' Copyright 2017, Gurobi Optimization, Inc.

' Assign workers to shifts; each worker may or may not be available on a
' particular day. If the problem cannot be solved, relax the model
' to determine which constraints cannot be satisfied, and how much
' they need to be relaxed.

Imports System
Imports Gurobi

Class workforce3_vb
    Shared Sub Main()
        Try

            ' Sample data
            ' Sets of days and workers
            Dim Shifts As String() = New String() {"Mon1", "Tue2", "Wed3", "Thu4", _
                                                   "Fri5", "Sat6", "Sun7", "Mon8", _
                                                   "Tue9", "Wed10", "Thu11", _
                                                   "Fri12", "Sat13", "Sun14"}
            Dim Workers As String() = New String() {"Amy", "Bob", "Cathy", "Dan", _
                                                    "Ed", "Fred", "Gu"}

            Dim nShifts As Integer = Shifts.Length
            Dim nWorkers As Integer = Workers.Length

            ' Number of workers required for each shift
            Dim shiftRequirements As Double() = New Double() {3, 2, 4, 4, 5, 6, _
                                                              5, 2, 2, 3, 4, 6, _
                                                              7, 5}

            ' Amount each worker is paid to work one shift
            Dim pay As Double() = New Double() {10, 12, 10, 8, 8, 9, 11}

            ' Worker availability: 0 if the worker is unavailable for a shift
            Dim availability As Double(,) = New Double(,) { _
                        {0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1}, _
                        {1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0}, _
                        {0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1}, _
                        {0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1}, _
                        {1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1}, _
                        {1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1}, _
                        {1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}}
```

```
' Model
Dim env As New GRBEnv()
Dim model As New GRBModel(env)

model.ModelName = "assignment"

' Assignment variables: x[w][s] == 1 if worker w is assigned
' to shift s. Since an assignment model always produces integer
' solutions, we use continuous variables and solve as an LP.
Dim x As GRBVar(,) = New GRBVar(nWorkers - 1, nShifts - 1) {}
For w As Integer = 0 To nWorkers - 1
    For s As Integer = 0 To nShifts - 1
        x(w, s) = model.AddVar(0, availability(w, s), pay(w), _
                               GRB.CONTINUOUS, _
                               Workers(w) & "." & Shifts(s))
    Next
Next

' The objective is to minimize the total pay costs
model.ModelSense = GRB.MINIMIZE

' Constraint: assign exactly shiftRequirements[s] workers
' to each shift s
For s As Integer = 0 To nShifts - 1
    Dim lhs As GRBLinExpr = 0.0
    For w As Integer = 0 To nWorkers - 1
        lhs.AddTerm(1.0, x(w, s))
    Next
    model.AddConstr(lhs = shiftRequirements(s), Shifts(s))
Next

' Optimize
model.Optimize()
Dim status As Integer = model.Status
If status = GRB.Status.UNBOUNDED Then
    Console.WriteLine("The model cannot be solved " & _
                      "because it is unbounded")
    Return
End If
If status = GRB.Status.OPTIMAL Then
    Console.WriteLine("The optimal objective is " & model.ObjVal)
    Return
End If
If (status <> GRB.Status.INF_OR_UNBD) AndAlso _
   (status <> GRB.Status.INFEASIBLE) Then
```

```
            Console.WriteLine("Optimization was stopped with status " & _
                              status)
            Return
        End If

        ' Relax the constraints to make the model feasible
        Console.WriteLine("The model is infeasible; relaxing the constraints")
        Dim orignumvars As Integer = model.NumVars
        model.FeasRelax(0, False, False, True)
        model.Optimize()
        status = model.Status
        If (status = GRB.Status.INF_OR_UNBD) OrElse _
           (status = GRB.Status.INFEASIBLE) OrElse _
           (status = GRB.Status.UNBOUNDED) Then
            Console.WriteLine("The relaxed model cannot be solved " & _
                              "because it is infeasible or unbounded")
            Return
        End If
        If status <> GRB.Status.OPTIMAL Then
            Console.WriteLine("Optimization was stopped with status " & status)
            Return
        End If

        Console.WriteLine(vbLf & "Slack values:")
        Dim vars As GRBVar() = model.GetVars()
        For i As Integer = orignumvars To model.NumVars - 1
            Dim sv As GRBVar = vars(i)
            If sv.X > 1E-06 Then
                Console.WriteLine(sv.VarName & " = " & sv.X)
            End If
        Next

        ' Dispose of model and environment
        model.Dispose()

        env.Dispose()
    Catch e As GRBException
        Console.WriteLine("Error code: " + e.ErrorCode & ". " + e.Message)
    End Try
  End Sub
End Class
```

workforce4_vb.vb

```
' Copyright 2017, Gurobi Optimization, Inc.
'
' Assign workers to shifts; each worker may or may not be available on a
' particular day. We use Pareto optimization to solve the model:
' first, we minimize the linear sum of the slacks. Then, we constrain
' the sum of the slacks, and we minimize a quadratic objective that
' tries to balance the workload among the workers.

Imports System
Imports Gurobi

Class workforce4_vb
    Shared Sub Main()
        Try

            ' Sample data
            ' Sets of days and workers
            Dim Shifts As String() = New String() {"Mon1", "Tue2", "Wed3", "Thu4", _
                                                   "Fri5", "Sat6", "Sun7", "Mon8", _
                                                   "Tue9", "Wed10", "Thu11", _
                                                   "Fri12", "Sat13", "Sun14"}
            Dim Workers As String() = New String() {"Amy", "Bob", "Cathy", "Dan", _
                                                    "Ed", "Fred", "Gu"}

            Dim nShifts As Integer = Shifts.Length
            Dim nWorkers As Integer = Workers.Length

            ' Number of workers required for each shift
            Dim shiftRequirements As Double() = New Double() {3, 2, 4, 4, 5, 6, _
                                                              5, 2, 2, 3, 4, 6, _
                                                              7, 5}

            ' Worker availability: 0 if the worker is unavailable for a shift
            Dim availability As Double(,) = New Double(,) { _
                      {0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1}, _
                      {1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0}, _
                      {0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1}, _
                      {0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1}, _
                      {1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1}, _
                      {1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1}, _
                      {1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}}

            ' Model
            Dim env As New GRBEnv()
```

```
Dim model As New GRBModel(env)

model.ModelName = "assignment"

' Assignment variables: x(w)(s) == 1 if worker w is assigned
' to shift s. This is no longer a pure assignment model, so we
' must use binary variables.
Dim x As GRBVar(,) = New GRBVar(nWorkers - 1, nShifts - 1) {}
For w As Integer = 0 To nWorkers - 1
    For s As Integer = 0 To nShifts - 1
        x(w, s) = model.AddVar(0, availability(w, s), 0, _
                               GRB.BINARY, _
                               Workers(w) & "." & Shifts(s))
    Next
Next

' Add a new slack variable to each shift constraint so that the
' shifts can be satisfied
Dim slacks As GRBVar() = New GRBVar(nShifts - 1) {}
For s As Integer = 0 To nShifts - 1
    slacks(s) = _
        model.AddVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS, _
                     Shifts(s) & "Slack")
Next

' Variable to represent the total slack
Dim totSlack As GRBVar = model.AddVar(0, GRB.INFINITY, 0, _
                                      GRB.CONTINUOUS, "totSlack")

' Variables to count the total shifts worked by each worker
Dim totShifts As GRBVar() = New GRBVar(nWorkers - 1) {}
For w As Integer = 0 To nWorkers - 1
    totShifts(w) = _
        model.AddVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS, _
                     Workers(w) & "TotShifts")
Next

Dim lhs As GRBLinExpr

' Constraint: assign exactly shiftRequirements(s) workers
' to each shift s, plus the slack
For s As Integer = 0 To nShifts - 1
    lhs = 0
    lhs.AddTerm(1.0, slacks(s))
    For w As Integer = 0 To nWorkers - 1
```

```
        lhs.AddTerm(1.0, x(w, s))
    Next
    model.AddConstr(lhs = shiftRequirements(s), Shifts(s))
Next

' Constraint: set totSlack equal to the total slack
lhs = 0
For s As Integer = 0 To nShifts - 1
    lhs.AddTerm(1.0, slacks(s))
Next
model.AddConstr(lhs = totSlack, "totSlack")

' Constraint: compute the total number of shifts for each worker
For w As Integer = 0 To nWorkers - 1
    lhs = 0
    For s As Integer = 0 To nShifts - 1
        lhs.AddTerm(1.0, x(w, s))
    Next
    model.AddConstr(lhs = totShifts(w), "totShifts" & Workers(w))
Next

' Objective: minimize the total slack
model.SetObjective(1.0*totSlack)

' Optimize
Dim status As Integer = _
    solveAndPrint(model, totSlack, nWorkers, Workers, totShifts)
If status <> GRB.Status.OPTIMAL Then
    Exit Sub
End If

' Constrain the slack by setting its upper and lower bounds
totSlack.UB = totSlack.X
totSlack.LB = totSlack.X

' Variable to count the average number of shifts worked
Dim avgShifts As GRBVar = model.AddVar(0, GRB.INFINITY, 0, _
                                       GRB.CONTINUOUS, "avgShifts")

' Variables to count the difference from average for each worker;
' note that these variables can take negative values.
Dim diffShifts As GRBVar() = New GRBVar(nWorkers - 1) {}
For w As Integer = 0 To nWorkers - 1
    diffShifts(w) = _
        model.AddVar(-GRB.INFINITY, GRB.INFINITY, 0, _
```

```
                            GRB.CONTINUOUS, Workers(w) & "Diff")
        Next

        ' Constraint: compute the average number of shifts worked
        lhs = 0
        For w As Integer = 0 To nWorkers - 1
            lhs.AddTerm(1.0, totShifts(w))
        Next
        model.AddConstr(lhs = nWorkers * avgShifts, "avgShifts")

        ' Constraint: compute the difference from the average number of shifts
        For w As Integer = 0 To nWorkers - 1
            model.AddConstr(totShifts(w) - avgShifts = diffShifts(w), _
                            Workers(w) & "Diff")
        Next

        ' Objective: minimize the sum of the square of the difference
        ' from the average number of shifts worked
        Dim qobj As GRBQuadExpr = New GRBQuadExpr
        For w As Integer = 0 To nWorkers - 1
            qobj.AddTerm(1.0, diffShifts(w), diffShifts(w))
        Next
        model.SetObjective(qobj)

        ' Optimize
        status = solveAndPrint(model, totSlack, nWorkers, Workers, totShifts)
        If status <> GRB.Status.OPTIMAL Then
            Exit Sub
        End If

        ' Dispose of model and env
        model.Dispose()
        env.Dispose()

    Catch e As GRBException
        Console.WriteLine("Error code: " & e.ErrorCode & ". " & e.Message)
    End Try
End Sub

Private Shared Function solveAndPrint(ByVal model As GRBModel, _
                                      ByVal totSlack As GRBVar, _
                                      ByVal nWorkers As Integer, _
                                      ByVal Workers As String(), _
                                      ByVal totShifts As GRBVar()) As Integer
    model.Optimize()
```

```
        Dim status As Integer = model.Status
        solveAndPrint = status
        If (status = GRB.Status.INF_OR_UNBD) OrElse _
           (status = GRB.Status.INFEASIBLE) OrElse _
           (status = GRB.Status.UNBOUNDED) Then
            Console.WriteLine("The model cannot be solved because " & _
                              "it is infeasible or unbounded")
            Exit Function
        End If
        If status <> GRB.Status.OPTIMAL Then
            Console.WriteLine("Optimization was stopped with status " _
                              & status)
            Exit Function
        End If

        ' Print total slack and the number of shifts worked for each worker
        Console.WriteLine(vbLf & "Total slack required: " & totSlack.X)
        For w As Integer = 0 To nWorkers - 1
            Console.WriteLine(Workers(w) & " worked " & _
                              totShifts(w).X & " shifts")
        Next

        Console.WriteLine(vbLf)
    End Function
End Class
```

workforce5_vb.vb

```
' Copyright 2017, Gurobi Optimization, Inc.
'
' Assign workers to shifts; each worker may or may not be available on a
' particular day. We use multi-objective optimization to solve the model.
' The highest-priority objective minimizes the sum of the slacks
' (i.e., the total number of uncovered shifts). The secondary objective
' minimizes the difference between the maximum and minimum number of
' shifts worked among all workers.  The second optimization is allowed
' to degrade the first objective by up to the smaller value of 10% and 2 */

Imports System
Imports Gurobi

Class workforce5_vb
    Shared Sub Main()

        Try

            ' Sample data
            ' Sets of days and workers
            Dim Shifts As String() = New String() { _
                "Mon1", "Tue2", "Wed3", "Thu4", "Fri5", "Sat6", "Sun7", _
                "Mon8", "Tue9", "Wed10", "Thu11", "Fri12", "Sat13", "Sun14"}

            Dim Workers As String() = New String() { _
                "Amy", "Bob", "Cathy", "Dan", "Ed", "Fred", "Gu", "Tobi"}

            Dim nShifts As Integer = Shifts.Length
            Dim nWorkers As Integer = Workers.Length

            ' Number of workers required for each shift
            Dim shiftRequirements As Double() = New Double() { _
                3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5}

            ' Worker availability: 0 if the worker is unavailable for a shift
            Dim availability As Double(,) = New Double(,) { _
                {0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1}, _
                {1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0}, _
                {0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1}, _
                {0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1}, _
                {1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1}, _
                {1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1}, _
                {0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1}, _
```

```
    {1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}}

' Create environment
Dim env As New GRBEnv()

' Create initial model
Dim model As New GRBModel(env)
model.ModelName = "workforce5_vb"

' Initialize assignment decision variables:
' x[w][s] == 1 if worker w is assigned to shift s.
' This is no longer a pure assignment model, so we must
' use binary variables.
Dim x As GRBVar(,) = New GRBVar(nWorkers - 1, nShifts - 1) {}
For w As Integer = 0 To nWorkers - 1
    For s As Integer = 0 To nShifts - 1
        x(w, s) = model.AddVar(0, availability(w, s), 0, GRB.BINARY, _
                               String.Format("{0}.{1}", Workers(w), Shifts(s)))
    Next
Next

' Slack variables for each shift constraint so that the shifts can
' be satisfied
Dim slacks As GRBVar() = New GRBVar(nShifts - 1) {}
For s As Integer = 0 To nShifts - 1
    slacks(s) = model.AddVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS, _
                             String.Format("{0}Slack", Shifts(s)))
Next

' Variable to represent the total slack
Dim totSlack As GRBVar = model.AddVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS, "totSlack"

' Variables to count the total shifts worked by each worker
Dim totShifts As GRBVar() = New GRBVar(nWorkers - 1) {}
For w As Integer = 0 To nWorkers - 1
    totShifts(w) = model.AddVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS, _
                                String.Format("{0}TotShifts", Workers(w)))
Next

Dim lhs As GRBLinExpr

' Constraint: assign exactly shiftRequirements[s] workers
' to each shift s, plus the slack
For s As Integer = 0 To nShifts - 1
    lhs = New GRBLinExpr()
```

```
    lhs.AddTerm(1.0, slacks(s))
    For w As Integer = 0 To nWorkers - 1
        lhs.AddTerm(1.0, x(w, s))
    Next
    model.AddConstr(lhs, GRB.EQUAL, shiftRequirements(s), Shifts(s))
Next

' Constraint: set totSlack equal to the total slack
lhs = New GRBLinExpr()
lhs.AddTerm(-1.0, totSlack)
For s As Integer = 0 To nShifts - 1
    lhs.AddTerm(1.0, slacks(s))
Next
model.AddConstr(lhs, GRB.EQUAL, 0, "totSlack")

' Constraint: compute the total number of shifts for each worker
For w As Integer = 0 To nWorkers - 1
    lhs = New GRBLinExpr()
    lhs.AddTerm(-1.0, totShifts(w))
    For s As Integer = 0 To nShifts - 1
        lhs.AddTerm(1.0, x(w, s))
    Next
    model.AddConstr(lhs, GRB.EQUAL, 0, String.Format("totShifts{0}", Workers(w)))
Next

' Constraint: set minShift/maxShift variable to less <=/>= to the
' number of shifts among all workers
Dim minShift As GRBVar = model.AddVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS, "minShift"
Dim maxShift As GRBVar = model.AddVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS, "maxShift"
model.AddGenConstrMin(minShift, totShifts, GRB.INFINITY, "minShift")
model.AddGenConstrMax(maxShift, totShifts, -GRB.INFINITY, "maxShift")

' Set global sense for ALL objectives
model.ModelSense = GRB.MINIMIZE

' Set primary objective
model.SetObjectiveN(totSlack, 0, 2, 1.0, 2.0, 0.1, "TotalSlack")

' Set secondary objective
model.SetObjectiveN(maxShift - minShift, 1, 1, 1.0, 0, 0, "Fairness")

' Save problem
model.Write("workforce5_vb.lp")

' Optimize
```

```
        Dim status As Integer = _
            solveAndPrint(model, totSlack, nWorkers, Workers, totShifts)

        If status <> GRB.Status.OPTIMAL Then
            Return
        End If

        ' Dispose of model and environment
        model.Dispose()

        env.Dispose()
    Catch e As GRBException
        Console.WriteLine("Error code: {0}. {1}", e.ErrorCode, e.Message)
    End Try
End Sub

Private Shared Function solveAndPrint(ByVal model As GRBModel, _
                                      ByVal totSlack As GRBVar, _
                                      ByVal nWorkers As Integer, _
                                      ByVal Workers As String(), _
                                      ByVal totShifts As GRBVar()) As Integer

    model.Optimize()
    Dim status As Integer = model.Status
    If status = GRB.Status.INF_OR_UNBD OrElse _
       status = GRB.Status.INFEASIBLE OrElse _
       status = GRB.Status.UNBOUNDED Then
        Console.WriteLine("The model cannot be solved " & _
                  "because it is infeasible or unbounded")
        Return status
    End If
    If status <> GRB.Status.OPTIMAL Then
        Console.WriteLine("Optimization was stopped with status {0}", status)
        Return status
    End If

    ' Print total slack and the number of shifts worked for each worker
    Console.WriteLine(vbLf & "Total slack required: {0}", totSlack.X)
    For w As Integer = 0 To nWorkers - 1
        Console.WriteLine("{0} worked {1} shifts", Workers(w), totShifts(w).X)
    Next
    Console.WriteLine(vbLf)
    Return status
End Function
```

```
End Class
```

3.6 Python Examples

This section includes source code for all of the Gurobi Python examples. The same source code can be found in the `examples/python` directory of the Gurobi distribution.

callback.py

```
#!/usr/bin/python

# Copyright 2017, Gurobi Optimization, Inc.

#   This example reads a model from a file, sets up a callback that
#   monitors optimization progress and implements a custom
#   termination strategy, and outputs progress information to the
#   screen and to a log file.
#
#   The termination strategy implemented in this callback stops the
#   optimization of a MIP model once at least one of the following two
#   conditions have been satisfied:
#     1) The optimality gap is less than 10%
#     2) At least 10000 nodes have been explored, and an integer feasible
#        solution has been found.
#   Note that termination is normally handled through Gurobi parameters
#   (MIPGap, NodeLimit, etc.).  You should only use a callback for
#   termination if the available parameters don't capture your desired
#   termination criterion.

import sys
from gurobipy import *

# Define my callback function

def mycallback(model, where):
    if where == GRB.Callback.POLLING:
        # Ignore polling callback
        pass
    elif where == GRB.Callback.PRESOLVE:
        # Presolve callback
        cdels = model.cbGet(GRB.Callback.PRE_COLDEL)
        rdels = model.cbGet(GRB.Callback.PRE_ROWDEL)
        if cdels or rdels:
            print('%d columns and %d rows are removed' % (cdels, rdels))
    elif where == GRB.Callback.SIMPLEX:
        # Simplex callback
        itcnt = model.cbGet(GRB.Callback.SPX_ITRCNT)
        if itcnt - model._lastiter >= 100:
```

```
            model._lastiter = itcnt
            obj = model.cbGet(GRB.Callback.SPX_OBJVAL)
            ispert = model.cbGet(GRB.Callback.SPX_ISPERT)
            pinf = model.cbGet(GRB.Callback.SPX_PRIMINF)
            dinf = model.cbGet(GRB.Callback.SPX_DUALINF)
            if ispert == 0:
                ch = ' '
            elif ispert == 1:
                ch = 'S'
            else:
                ch = 'P'
            print('%d %g%s %g %g' % (int(itcnt), obj, ch, pinf, dinf))
    elif where == GRB.Callback.MIP:
        # General MIP callback
        nodecnt = model.cbGet(GRB.Callback.MIP_NODCNT)
        objbst = model.cbGet(GRB.Callback.MIP_OBJBST)
        objbnd = model.cbGet(GRB.Callback.MIP_OBJBND)
        solcnt = model.cbGet(GRB.Callback.MIP_SOLCNT)
        if nodecnt - model._lastnode >= 100:
            model._lastnode = nodecnt
            actnodes = model.cbGet(GRB.Callback.MIP_NODLFT)
            itcnt = model.cbGet(GRB.Callback.MIP_ITRCNT)
            cutcnt = model.cbGet(GRB.Callback.MIP_CUTCNT)
            print('%d %d %d %g %g %d %d' % (nodecnt, actnodes, \
                  itcnt, objbst, objbnd, solcnt, cutcnt))
        if abs(objbst - objbnd) < 0.1 * (1.0 + abs(objbst)):
            print('Stop early - 10% gap achieved')
            model.terminate()
        if nodecnt >= 10000 and solcnt:
            print('Stop early - 10000 nodes explored')
            model.terminate()
    elif where == GRB.Callback.MIPSOL:
        # MIP solution callback
        nodecnt = model.cbGet(GRB.Callback.MIPSOL_NODCNT)
        obj = model.cbGet(GRB.Callback.MIPSOL_OBJ)
        solcnt = model.cbGet(GRB.Callback.MIPSOL_SOLCNT)
        x = model.cbGetSolution(model._vars)
        print('**** New solution at node %d, obj %g, sol %d, ' \
              'x[0] = %g ****' % (nodecnt, obj, solcnt, x[0]))
    elif where == GRB.Callback.MIPNODE:
        # MIP node callback
        print('**** New node ****')
        if model.cbGet(GRB.Callback.MIPNODE_STATUS) == GRB.Status.OPTIMAL:
            x = model.cbGetNodeRel(model._vars)
            model.cbSetSolution(model.getVars(), x)
```

```
    elif where == GRB.Callback.BARRIER:
        # Barrier callback
        itcnt = model.cbGet(GRB.Callback.BARRIER_ITRCNT)
        primobj = model.cbGet(GRB.Callback.BARRIER_PRIMOBJ)
        dualobj = model.cbGet(GRB.Callback.BARRIER_DUALOBJ)
        priminf = model.cbGet(GRB.Callback.BARRIER_PRIMINF)
        dualinf = model.cbGet(GRB.Callback.BARRIER_DUALINF)
        cmpl = model.cbGet(GRB.Callback.BARRIER_COMPL)
        print('%d %g %g %g %g %g' % (itcnt, primobj, dualobj, \
              priminf, dualinf, cmpl))
    elif where == GRB.Callback.MESSAGE:
        # Message callback
        msg = model.cbGet(GRB.Callback.MSG_STRING)
        model._logfile.write(msg)


if len(sys.argv) < 2:
    print('Usage: callback.py filename')
    quit()

# Turn off display and heuristics

setParam('OutputFlag', 0)
setParam('Heuristics', 0)

# Read model from file

model = read(sys.argv[1])

# Open log file

logfile = open('cb.log', 'w')

# Pass data into my callback function

model._lastiter = -GRB.INFINITY
model._lastnode = -GRB.INFINITY
model._logfile = logfile
model._vars = model.getVars()

# Solve model and capture solution information

model.optimize(mycallback)

print('')
```

```
print('Optimization complete')
if model.SolCount == 0:
    print('No solution found, optimization status = %d' % model.Status)
else:
    print('Solution found, objective = %g' % model.ObjVal)
    for v in model.getVars():
        if v.X != 0.0:
            print('%s %g' % (v.VarName, v.X))

# Close log file

logfile.close()
```

custom.py

```
#
# Copyright 2017, Gurobi Optimization, Inc.
#
# Interactive shell customization example
#
# Define a set of customizations for the Gurobi shell.
# Type 'from custom import *' to import them into your shell.
#

from gurobipy import *


# custom read command --- change directory as appropriate

def myread(name):
    return read('/home/jones/models/' + name)


# simple termination callback

def mycallback(model, where):
    if where == GRB.Callback.MIP:
        time = model.cbGet(GRB.Callback.RUNTIME)
        best = model.cbGet(GRB.Callback.MIP_OBJBST)
        if time > 10 and best < GRB.INFINITY:
            model.terminate()


# custom optimize() function that uses callback

def myopt(model):
    model.optimize(mycallback)
```

dense.py

```
#!/usr/bin/python

# Copyright 2017, Gurobi Optimization, Inc.

# This example formulates and solves the following simple QP model:
#
#    minimize    x + y + x^2 + x*y + y^2 + y*z + z^2
#    subject to  x + 2 y + 3 z >= 4
#                x +   y       >= 1
#
# The example illustrates the use of dense matrices to store A and Q
# (and dense vectors for the other relevant data).  We don't recommend
# that you use dense matrices, but this example may be helpful if you
# already have your data in this format.

import sys
from gurobipy import *

def dense_optimize(rows, cols, c, Q, A, sense, rhs, lb, ub, vtype,
                   solution):

  model = Model()

  # Add variables to model
  vars = []
  for j in range(cols):
    vars.append(model.addVar(lb=lb[j], ub=ub[j], vtype=vtype[j]))

  # Populate A matrix
  for i in range(rows):
    expr = LinExpr()
    for j in range(cols):
      if A[i][j] != 0:
        expr += A[i][j]*vars[j]
    model.addConstr(expr, sense[i], rhs[i])

  # Populate objective
  obj = QuadExpr()
  for i in range(cols):
    for j in range(cols):
      if Q[i][j] != 0:
        obj += Q[i][j]*vars[i]*vars[j]
  for j in range(cols):
    if c[j] != 0:
```

```
      obj += c[j]*vars[j]
  model.setObjective(obj)

  # Solve
  model.optimize()

  # Write model to a file
  model.write('dense.lp')

  if model.status == GRB.Status.OPTIMAL:
    x = model.getAttr('x', vars)
    for i in range(cols):
      solution[i] = x[i]
    return True
  else:
    return False


# Put model data into dense matrices

c = [1, 1, 0]
Q = [[1, 1, 0], [0, 1, 1], [0, 0, 1]]
A = [[1, 2, 3], [1, 1, 0]]
sense = [GRB.GREATER_EQUAL, GRB.GREATER_EQUAL]
rhs = [4, 1]
lb = [0, 0, 0]
ub = [GRB.INFINITY, GRB.INFINITY, GRB.INFINITY]
vtype = [GRB.CONTINUOUS, GRB.CONTINUOUS, GRB.CONTINUOUS]
sol = [0]*3

# Optimize

success = dense_optimize(2, 3, c, Q, A, sense, rhs, lb, ub, vtype, sol)

if success:
  print('x: %g, y: %g, z: %g' % (sol[0], sol[1], sol[2]))
```

diet.py

```
#!/usr/bin/python

# Copyright 2017, Gurobi Optimization, Inc.

# Solve the classic diet model, showing how to add constraints
# to an existing model.

from gurobipy import *

# Nutrition guidelines, based on
# USDA Dietary Guidelines for Americans, 2005
# http://www.health.gov/DietaryGuidelines/dga2005/

categories, minNutrition, maxNutrition = multidict({
  'calories': [1800, 2200],
  'protein':  [91, GRB.INFINITY],
  'fat':      [0, 65],
  'sodium':   [0, 1779] })

foods, cost = multidict({
  'hamburger': 2.49,
  'chicken':   2.89,
  'hot dog':   1.50,
  'fries':     1.89,
  'macaroni':  2.09,
  'pizza':     1.99,
  'salad':     2.49,
  'milk':      0.89,
  'ice cream': 1.59 })

# Nutrition values for the foods
nutritionValues = {
  ('hamburger', 'calories'): 410,
  ('hamburger', 'protein'):  24,
  ('hamburger', 'fat'):      26,
  ('hamburger', 'sodium'):   730,
  ('chicken',   'calories'): 420,
  ('chicken',   'protein'):  32,
  ('chicken',   'fat'):      10,
  ('chicken',   'sodium'):   1190,
  ('hot dog',   'calories'): 560,
  ('hot dog',   'protein'):  20,
  ('hot dog',   'fat'):      32,
  ('hot dog',   'sodium'):   1800,
```

```
  ('fries',     'calories'): 380,
  ('fries',     'protein'):  4,
  ('fries',     'fat'):      19,
  ('fries',     'sodium'):   270,
  ('macaroni',  'calories'): 320,
  ('macaroni',  'protein'):  12,
  ('macaroni',  'fat'):      10,
  ('macaroni',  'sodium'):   930,
  ('pizza',     'calories'): 320,
  ('pizza',     'protein'):  15,
  ('pizza',     'fat'):      12,
  ('pizza',     'sodium'):   820,
  ('salad',     'calories'): 320,
  ('salad',     'protein'):  31,
  ('salad',     'fat'):      12,
  ('salad',     'sodium'):   1230,
  ('milk',      'calories'): 100,
  ('milk',      'protein'):  8,
  ('milk',      'fat'):      2.5,
  ('milk',      'sodium'):   125,
  ('ice cream', 'calories'): 330,
  ('ice cream', 'protein'):  8,
  ('ice cream', 'fat'):      10,
  ('ice cream', 'sodium'):   180 }

# Model
m = Model("diet")

# Create decision variables for the foods to buy
buy = m.addVars(foods, name="buy")

# You could use Python looping constructs and m.addVar() to create
# these decision variables instead.  The following would be equivalent
#
# buy = {}
# for f in foods:
#   buy[f] = m.addVar(name=f)

# The objective is to minimize the costs
m.setObjective(buy.prod(cost), GRB.MINIMIZE)

# Using looping constructs, the preceding statement would be:
#
# m.setObjective(sum(buy[f]*cost[f] for f in foods), GRB.MINIMIZE)
```

```
# Nutrition constraints
m.addConstrs(
    (quicksum(nutritionValues[f,c] * buy[f] for f in foods)
            == [minNutrition[c], maxNutrition[c]]
     for c in categories), "_")

# Using looping constructs, the preceding statement would be:
#
# for c in categories:
#  m.addRange(
#     sum(nutritionValues[f,c] * buy[f] for f in foods), minNutrition[c], maxNutrition[c], c)

def printSolution():
    if m.status == GRB.Status.OPTIMAL:
        print('\nCost: %g' % m.objVal)
        print('\nBuy:')
        buyx = m.getAttr('x', buy)
        for f in foods:
            if buy[f].x > 0.0001:
                print('%s %g' % (f, buyx[f]))
    else:
        print('No solution')

# Solve
m.optimize()
printSolution()

print('\nAdding constraint: at most 6 servings of dairy')
m.addConstr(buy.sum(['milk','ice cream']) <= 6, "limit_dairy")

# Solve
m.optimize()
printSolution()
```

diet2.py

```
#!/usr/bin/python

# Copyright 2017, Gurobi Optimization, Inc.

# Separate the model (dietmodel.py) from the data file (diet2.py), so
# that the model can be solved with different data files.
#
# Nutrition guidelines, based on
# USDA Dietary Guidelines for Americans, 2005
# http://www.health.gov/DietaryGuidelines/dga2005/

from gurobipy import *

categories, minNutrition, maxNutrition = multidict({
  'calories': [1800, 2200],
  'protein':  [91, GRB.INFINITY],
  'fat':      [0, 65],
  'sodium':   [0, 1779] })

foods, cost = multidict({
  'hamburger': 2.49,
  'chicken':   2.89,
  'hot dog':   1.50,
  'fries':     1.89,
  'macaroni':  2.09,
  'pizza':     1.99,
  'salad':     2.49,
  'milk':      0.89,
  'ice cream': 1.59 })

# Nutrition values for the foods
nutritionValues = {
  ('hamburger', 'calories'): 410,
  ('hamburger', 'protein'):  24,
  ('hamburger', 'fat'):      26,
  ('hamburger', 'sodium'):   730,
  ('chicken',   'calories'): 420,
  ('chicken',   'protein'):  32,
  ('chicken',   'fat'):      10,
  ('chicken',   'sodium'):   1190,
  ('hot dog',   'calories'): 560,
  ('hot dog',   'protein'):  20,
  ('hot dog',   'fat'):      32,
  ('hot dog',   'sodium'):   1800,
```

```
  ('fries',     'calories'): 380,
  ('fries',     'protein'):  4,
  ('fries',     'fat'):      19,
  ('fries',     'sodium'):   270,
  ('macaroni',  'calories'): 320,
  ('macaroni',  'protein'):  12,
  ('macaroni',  'fat'):      10,
  ('macaroni',  'sodium'):   930,
  ('pizza',     'calories'): 320,
  ('pizza',     'protein'):  15,
  ('pizza',     'fat'):      12,
  ('pizza',     'sodium'):   820,
  ('salad',     'calories'): 320,
  ('salad',     'protein'):  31,
  ('salad',     'fat'):      12,
  ('salad',     'sodium'):   1230,
  ('milk',      'calories'): 100,
  ('milk',      'protein'):  8,
  ('milk',      'fat'):      2.5,
  ('milk',      'sodium'):   125,
  ('ice cream', 'calories'): 330,
  ('ice cream', 'protein'):  8,
  ('ice cream', 'fat'):      10,
  ('ice cream', 'sodium'):   180 }

import dietmodel
dietmodel.solve(categories, minNutrition, maxNutrition,
                foods, cost, nutritionValues)
```

diet3.py

```
#!/usr/bin/python

# Copyright 2017, Gurobi Optimization, Inc.

# Use a SQLite database with the diet model (dietmodel.py).  The database
# (diet.db) can be recreated using the included SQL script (diet.sql).
#
# Note that this example reads an external data file (..\data\diet.db).
# As a result, it must be run from the Gurobi examples/python directory.

import os
import sqlite3
from gurobipy import *

con = sqlite3.connect(os.path.join('..', 'data', 'diet.db'))
cur = con.cursor()

cur.execute('select category,minnutrition,maxnutrition from categories')
result = cur.fetchall()
categories, minNutrition, maxNutrition = multidict(
    (cat,[minv,maxv]) for cat,minv,maxv in result)

cur.execute('select food,cost from foods')
result = cur.fetchall()
foods, cost = multidict(result)

cur.execute('select food,category,value from nutrition')
result = cur.fetchall()
nutritionValues = dict(((f,c),v) for f,c,v in result)

con.close()

import dietmodel
dietmodel.solve(categories, minNutrition, maxNutrition,
                foods, cost, nutritionValues)
```

diet4.py

```
#!/usr/bin/python

# Copyright 2017, Gurobi Optimization, Inc.

# Read diet model data from an Excel spreadsheet (diet.xls).
# Pass the imported data into the diet model (dietmodel.py).
#
# Note that this example reads an external data file (..\data\diet.xls).
# As a result, it must be run from the Gurobi examples/python directory.
#
# This example requires Python package 'xlrd', which isn't included
# in most Python distributions.  You can obtain it from
# http://pypi.python.org/pypi/xlrd.

import os
import xlrd

book = xlrd.open_workbook(os.path.join("..", "data", "diet.xls"))

sh = book.sheet_by_name("Categories")
categories = []
minNutrition = {}
maxNutrition = {}
i = 1
while True:
    try:
        c = sh.cell_value(i, 0)
        categories.append(c)
        minNutrition[c] = sh.cell_value(i,1)
        maxNutrition[c] = sh.cell_value(i,2)
        i = i + 1
    except IndexError:
        break

sh = book.sheet_by_name("Foods")
foods = []
cost = {}
i = 1
while True:
    try:
        f = sh.cell_value(i, 0)
        foods.append(f)
        cost[f] = sh.cell_value(i,1)
        i = i + 1
```

```
    except IndexError:
        break

sh = book.sheet_by_name("Nutrition")
nutritionValues = {}
i = 1
for food in foods:
    j = 1
    for cat in categories:
        nutritionValues[food,cat] = sh.cell_value(i,j)
        j += 1
    i += 1

import dietmodel
dietmodel.solve(categories, minNutrition, maxNutrition,
                foods, cost, nutritionValues)
```

dietmodel.py

```
#!/usr/bin/python

# Copyright 2017, Gurobi Optimization, Inc.

# Solve the classic diet model.  This file implements
# a function that formulates and solves the model,
# but it contains no model data.  The data is
# passed in by the calling program.  Run example 'diet2.py',
# 'diet3.py', or 'diet4.py' to invoke this function.

from gurobipy import *


def solve(categories, minNutrition, maxNutrition, foods, cost,
          nutritionValues):
    # Model
    m = Model("diet")

    # Create decision variables for the foods to buy
    buy = m.addVars(foods, name="buy")

    # The objective is to minimize the costs
    m.setObjective(buy.prod(cost), GRB.MINIMIZE)

    # Nutrition constraints
    m.addConstrs(
        (quicksum(nutritionValues[f,c] * buy[f] for f in foods)
                == [minNutrition[c], maxNutrition[c]]
         for c in categories), "_")


    def printSolution():
        if m.status == GRB.Status.OPTIMAL:
            print('\nCost: %g' % m.objVal)
            print('\nBuy:')
            buyx = m.getAttr('x', buy)
            for f in foods:
                if buy[f].x > 0.0001:
                    print('%s %g' % (f, buyx[f]))
        else:
            print('No solution')

    # Solve
    m.optimize()
```

```
printSolution()

print('\nAdding constraint: at most 6 servings of dairy')
m.addConstr(buy.sum(['milk','ice cream']) <= 6, "limit_dairy")

# Solve
m.optimize()
printSolution()
```

facility.py

```
#!/usr/bin/python

# Copyright 2017, Gurobi Optimization, Inc.

# Facility location: a company currently ships its product from 5 plants
# to 4 warehouses. It is considering closing some plants to reduce
# costs. What plant(s) should the company close, in order to minimize
# transportation and fixed costs?
#
# Note that this example uses lists instead of dictionaries.  Since
# it does not work with sparse data, lists are a reasonable option.
#
# Based on an example from Frontline Systems:
#   http://www.solver.com/disfacility.htm
# Used with permission.

from gurobipy import *

# Warehouse demand in thousands of units
demand = [15, 18, 14, 20]

# Plant capacity in thousands of units
capacity = [20, 22, 17, 19, 18]

# Fixed costs for each plant
fixedCosts = [12000, 15000, 17000, 13000, 16000]

# Transportation costs per thousand units
transCosts = [[4000, 2000, 3000, 2500, 4500],
              [2500, 2600, 3400, 3000, 4000],
              [1200, 1800, 2600, 4100, 3000],
              [2200, 2600, 3100, 3700, 3200]]

# Range of plants and warehouses
plants = range(len(capacity))
warehouses = range(len(demand))

# Model
m = Model("facility")

# Plant open decision variables: open[p] == 1 if plant p is open.
open = m.addVars(plants,
                 vtype=GRB.BINARY,
                 obj=fixedCosts,
```

```
                name="open")

# Transportation decision variables: transport[w,p] captures the
# optimal quantity to transport to warehouse w from plant p
transport = m.addVars(warehouses, plants, obj=transCosts, name="trans")

# You could use Python looping constructs and m.addVar() to create
# these decision variables instead.  The following would be equivalent
# to the preceding two statements...
#
#open = []
#for p in plants:
#  open.append(m.addVar(vtype=GRB.BINARY,
#                       obj=fixedCosts[p],
#                       name="open[%d]" % p))
#
#transport = []
#for w in warehouses:
#  transport.append([])
#  for p in plants:
#    transport[w].append(m.addVar(obj=transCosts[w][p],
#                                 name="trans[%d,%d]" % (w, p)))

# The objective is to minimize the total fixed and variable costs
m.modelSense = GRB.MINIMIZE

# Production constraints
# Note that the right-hand limit sets the production to zero if the plant
# is closed
m.addConstrs(
    (transport.sum('*',p) <= capacity[p]*open[p] for p in plants),
    "Capacity")

# Using Python looping constructs, the preceding would be...
#
#for p in plants:
#  m.addConstr(sum(transport[w][p] for w in warehouses) <= capacity[p] * open[p],
#              "Capacity[%d]" % p)

# Demand constraints
m.addConstrs(
    (transport.sum(w) == demand[w] for w in warehouses),
    "Demand")

# ... and the preceding would be ...
```

```
#for w in warehouses:
#  m.addConstr(sum(transport[w][p] for p in plants) == demand[w], "Demand[%d]" % w)

# Guess at the starting point: close the plant with the highest fixed costs;
# open all others

# First, open all plants
for p in plants:
    open[p].start = 1.0

# Now close the plant with the highest fixed cost
print('Initial guess:')
maxFixed = max(fixedCosts)
for p in plants:
    if fixedCosts[p] == maxFixed:
        open[p].start = 0.0
        print('Closing plant %s' % p)
        break
print('')

# Use barrier to solve root relaxation
m.Params.method = 2

# Solve
m.optimize()

# Print solution
print('\nTOTAL COSTS: %g' % m.objVal)
print('SOLUTION:')
for p in plants:
    if open[p].x > 0.99:
        print('Plant %s open' % p)
        for w in warehouses:
            if transport[w,p].x > 0:
                print('  Transport %g units to warehouse %s' % \
                      (transport[w,p].x, w))
    else:
        print('Plant %s closed!' % p)
```

feasopt.py

```
#!/usr/bin/python

# Copyright 2017, Gurobi Optimization, Inc.

# This example reads a MIP model from a file, adds artificial
# variables to each constraint, and then minimizes the sum of the
# artificial variables.  A solution with objective zero corresponds
# to a feasible solution to the input model.
#
# We can also use FeasRelax feature to do it. In this example, we
# use minrelax=1, i.e. optimizing the returned model finds a solution
# that minimizes the original objective, but only from among those
# solutions that minimize the sum of the artificial variables.

import sys
from gurobipy import *

if len(sys.argv) < 2:
    print('Usage: feasopt.py filename')
    quit()

feasmodel = gurobi.read(sys.argv[1])

#create a copy to use FeasRelax feature later

feasmodel1 = feasmodel.copy()

# clear objective

feasmodel.setObjective(0.0)

# add slack variables

for c in feasmodel.getConstrs():
    sense = c.sense
    if sense != '>':
        feasmodel.addVar(obj=1.0, name="ArtN_" + c.constrName,
                         column=Column([-1], [c]))
    if sense != '<':
        feasmodel.addVar(obj=1.0, name="ArtP_" + c.constrName,
                         column=Column([1], [c]))

# optimize modified model
```

```
feasmodel.optimize()

feasmodel.write('feasopt.lp')

# use FeasRelax feature

feasmodel1.feasRelaxS(0, True, False, True);

feasmodel1.write("feasopt1.lp");

feasmodel1.optimize();
```

fixanddive.py

```
#!/usr/bin/python

# Copyright 2017, Gurobi Optimization, Inc.

# Implement a simple MIP heuristic.  Relax the model,
# sort variables based on fractionality, and fix the 25% of
# the fractional variables that are closest to integer variables.
# Repeat until either the relaxation is integer feasible or
# linearly infeasible.

import sys
from gurobipy import *


# Key function used to sort variables based on relaxation fractionality

def sortkey(v1):
    sol = v1.x
    return abs(sol-int(sol+0.5))


if len(sys.argv) < 2:
    print('Usage: fixanddive.py filename')
    quit()

# Read model

model = gurobi.read(sys.argv[1])

# Collect integer variables and relax them
intvars = []
for v in model.getVars():
    if v.vType != GRB.CONTINUOUS:
        intvars += [v]
        v.vType = GRB.CONTINUOUS

model.Params.outputFlag = 0

model.optimize()


# Perform multiple iterations.  In each iteration, identify the first
# quartile of integer variables that are closest to an integer value in the
# relaxation, fix them to the nearest integer, and repeat.
```

```
for iter in range(1000):

# create a list of fractional variables, sorted in order of increasing
# distance from the relaxation solution to the nearest integer value

    fractional = []
    for v in intvars:
        sol = v.x
        if abs(sol - int(sol+0.5)) > 1e-5:
            fractional += [v]

    fractional.sort(key=sortkey)

    print('Iteration %d, obj %g, fractional %d' % \
          (iter, model.objVal, len(fractional)))

    if len(fractional) == 0:
        print('Found feasible solution - objective %g' % model.objVal)
        break


# Fix the first quartile to the nearest integer value
    nfix = max(int(len(fractional)/4), 1)
    for i in range(nfix):
        v = fractional[i]
        fixval = int(v.x+0.5)
        v.lb = fixval
        v.ub = fixval
        print('  Fix %s to %g (rel %g)' % (v.varName, fixval, v.x))

    model.optimize()

# Check optimization result

    if model.status != GRB.Status.OPTIMAL:
        print('Relaxation is infeasible')
        break
```

genconstr.py

```
#!/usr/bin/python

# Copyright 2017, Gurobi Optimization, Inc.

# In this example we show the use of general constraints for modeling
# some common expressions. We use as an example a SAT-problem where we
# want to see if it is possible to satisfy at least four (or all) clauses
# of the logical for
#
# L = (x0 or ~x1 or x2)  and (x1 or ~x2 or x3)  and
#     (x2 or ~x3 or x0)  and (x3 or ~x0 or x1)  and
#     (~x0 or ~x1 or x2) and (~x1 or ~x2 or x3) and
#     (~x2 or ~x3 or x0) and (~x3 or ~x0 or x1)
#
# We do this by introducing two variables for each literal (itself and its
# negated value), a variable for each clause, and then two
# variables for indicating if we can satisfy four, and another to identify
# the minimum of the clauses (so if it is one, we can satisfy all clauses)
# and put these two variables in the objective.
# i.e. the Objective function will be
#
# maximize Obj0 + Obj1
#
#  Obj0 = MIN(Clause1, ... , Clause8)
#  Obj1 = 1 -> Clause1 + ... + Clause8 >= 4
#
# thus, the objective value will be two if and only if we can satisfy all
# clauses; one if and only if at least four clauses can be satisfied, and
# zero otherwise.

from gurobipy import *

try:
    NLITERALS = 4

    n = NLITERALS

    # Example data:
    #   e.g. {0, n+1, 2} means clause (x0 or ~x1 or x2)
    Clauses = [[  0, n+1, 2],
               [  1, n+2, 3],
               [  2, n+3, 0],
               [  3, n+0, 1],
               [n+0, n+1, 2],
```

```
            [n+1, n+2, 3],
            [n+2, n+3, 0],
            [n+3, n+0, 1]]

# Create a new model
model = Model("Genconstr")

# initialize decision variables and objective
Lit = model.addVars(NLITERALS, vtype=GRB.BINARY, name="X")
NotLit = model.addVars(NLITERALS, vtype=GRB.BINARY, name="NotX")

Cla = model.addVars(len(Clauses), vtype=GRB.BINARY, name="Clause")

Obj0 = model.addVar(vtype=GRB.BINARY, name="Obj0")
Obj1 = model.addVar(vtype=GRB.BINARY, name="Obj1")

# Link Xi and notXi
model.addConstrs((Lit[i] + NotLit[i] == 1.0 for i in range(NLITERALS)),
                 name="CNSTR_X")

# Link clauses and literals
for i, c in enumerate(Clauses):
    clause = []
    for l in c:
      if l >= n:
          clause.append(NotLit[l-n])
      else:
          clause.append(Lit[l])
    model.addConstr(Cla[i] == or_(clause), "CNSTR_Clause" + str(i))

# Link objs with clauses
model.addConstr(Obj0 == min_(Cla), name="CNSTR_Obj0")
model.addConstr((Obj1 == 1) >> (Cla.sum() >= 4.0), name="CNSTR_Obj1")

# Set optimization objective
model.setObjective(Obj0 + Obj1, GRB.MAXIMIZE)

# Save problem
model.write("genconstr.mps")
model.write("genconstr.lp")

# Optimize
model.optimize()

# Status checking
```

```
    status = model.getAttr(GRB.Attr.Status)

    if status == GRB.INF_OR_UNBD or \
       status == GRB.INFEASIBLE  or \
       status == GRB.UNBOUNDED:
        print("The model cannot be solved because it is infeasible or unbounded")
        sys.exit(1)
    if status != GRB.OPTIMAL:
        print("Optimization was stopped with status ", status)
        sys.exit(1)

    # Print result
    objval = model.getAttr(GRB.Attr.ObjVal)

    if objval > 1.9:
        print("Logical expression is satisfiable")
    elif objval > 0.9:
        print("At least four clauses can be satisfied")
    else:
        print("Not even three clauses can be satisfied")

except GurobiError as e:
    print('Error code ' + str(e.errno) + ": " + str(e))

except AttributeError:
    print('Encountered an attribute error')
```

lp.py

```
#!/usr/bin/python

# Copyright 2017, Gurobi Optimization, Inc.

# This example reads an LP model from a file and solves it.
# If the model is infeasible or unbounded, the example turns off
# presolve and solves the model again. If the model is infeasible,
# the example computes an Irreducible Inconsistent Subsystem (IIS),
# and writes it to a file

import sys
from gurobipy import *

if len(sys.argv) < 2:
    print('Usage: lp.py filename')
    quit()

# Read and solve model

model = read(sys.argv[1])
model.optimize()

if model.status == GRB.Status.INF_OR_UNBD:
    # Turn presolve off to determine whether model is infeasible
    # or unbounded
    model.setParam(GRB.Param.Presolve, 0)
    model.optimize()

if model.status == GRB.Status.OPTIMAL:
    print('Optimal objective: %g' % model.objVal)
    model.write('model.sol')
    exit(0)
elif model.status != GRB.Status.INFEASIBLE:
    print('Optimization was stopped with status %d' % model.status)
    exit(0)


# Model is infeasible - compute an Irreducible Inconsistent Subsystem (IIS)

print('')
print('Model is infeasible')
model.computeIIS()
model.write("model.ilp")
print("IIS written to file 'model.ilp'")
```

lpmethod.py

```
#!/usr/bin/python

# Copyright 2017, Gurobi Optimization, Inc.

# Solve a model with different values of the Method parameter;
# show which value gives the shortest solve time.

import sys
from gurobipy import *

if len(sys.argv) < 2:
    print('Usage: lpmethod.py filename')
    quit()

# Read model
m = read(sys.argv[1])

# Solve the model with different values of Method
bestTime = m.Params.timeLimit
bestMethod = -1
for i in range(3):
    m.reset()
    m.Params.method = i
    m.optimize()
    if m.status == GRB.Status.OPTIMAL:
        bestTime = m.Runtime
        bestMethod = i
        # Reduce the TimeLimit parameter to save time with other methods
        m.Params.timeLimit = bestTime

# Report which method was fastest
if bestMethod == -1:
    print('Unable to solve this model')
else:
    print('Solved in %g seconds with Method %d' % (bestTime, bestMethod))
```

lpmod.py

```
#!/usr/bin/python

# Copyright 2017, Gurobi Optimization, Inc.

# This example reads an LP model from a file and solves it.
# If the model can be solved, then it finds the smallest positive variable,
# sets its upper bound to zero, and resolves the model two ways:
# first with an advanced start, then without an advanced start
# (i.e. 'from scratch').

import sys
from gurobipy import *

if len(sys.argv) < 2:
    print('Usage: lpmod.py filename')
    quit()

# Read model and determine whether it is an LP

model = read(sys.argv[1])
if model.isMIP == 1:
    print('The model is not a linear program')
    exit(1)

model.optimize()

status = model.status

if status == GRB.Status.INF_OR_UNBD or status == GRB.Status.INFEASIBLE \
   or status == GRB.Status.UNBOUNDED:
    print('The model cannot be solved because it is infeasible or unbounded')
    exit(1)

if status != GRB.Status.OPTIMAL:
    print('Optimization was stopped with status %d' % status)
    exit(0)

# Find the smallest variable value
minVal = GRB.INFINITY
for v in model.getVars():
    if v.x > 0.0001 and v.x < minVal and v.lb == 0.0:
        minVal = v.x
        minVar = v
```

```
print('\n*** Setting %s from %g to zero ***\n' % (minVar.varName, minVal))
minVar.ub = 0.0

# Solve from this starting point
model.optimize()

# Save iteration & time info
warmCount = model.IterCount
warmTime = model.Runtime

# Reset the model and resolve
print('\n*** Resetting and solving without an advanced start ***\n')
model.reset()
model.optimize()

coldCount = model.IterCount
coldTime = model.Runtime

print('')
print('*** Warm start: %g iterations, %g seconds' % (warmCount, warmTime))
print('*** Cold start: %g iterations, %g seconds' % (coldCount, coldTime))
```

mip1.py

```
#!/usr/bin/python

# Copyright 2017, Gurobi Optimization, Inc.

# This example formulates and solves the following simple MIP model:
#  maximize
#        x +   y + 2 z
#  subject to
#        x + 2 y + 3 z <= 4
#        x +   y       >= 1
#  x, y, z binary

from gurobipy import *

try:

    # Create a new model
    m = Model("mip1")

    # Create variables
    x = m.addVar(vtype=GRB.BINARY, name="x")
    y = m.addVar(vtype=GRB.BINARY, name="y")
    z = m.addVar(vtype=GRB.BINARY, name="z")

    # Set objective
    m.setObjective(x + y + 2 * z, GRB.MAXIMIZE)

    # Add constraint: x + 2 y + 3 z <= 4
    m.addConstr(x + 2 * y + 3 * z <= 4, "c0")

    # Add constraint: x + y >= 1
    m.addConstr(x + y >= 1, "c1")

    m.optimize()

    for v in m.getVars():
        print('%s %g' % (v.varName, v.x))

    print('Obj: %g' % m.objVal)

except GurobiError as e:
    print('Error code ' + str(e.errno) + ": " + str(e))
```

```
except AttributeError:
    print('Encountered an attribute error')
```

mip2.py

```
#!/usr/bin/python

# Copyright 2017, Gurobi Optimization, Inc.

# This example reads a MIP model from a file, solves it and prints
# the objective values from all feasible solutions generated while
# solving the MIP. Then it creates the associated fixed model and
# solves that model.

import sys
from gurobipy import *

if len(sys.argv) < 2:
    print('Usage: mip2.py filename')
    quit()

# Read and solve model

model = read(sys.argv[1])

if model.isMIP == 0:
    print('Model is not a MIP')
    exit(0)

model.optimize()

if model.status == GRB.Status.OPTIMAL:
    print('Optimal objective: %g' % model.objVal)
elif model.status == GRB.Status.INF_OR_UNBD:
    print('Model is infeasible or unbounded')
    exit(0)
elif model.status == GRB.Status.INFEASIBLE:
    print('Model is infeasible')
    exit(0)
elif model.status == GRB.Status.UNBOUNDED:
    print('Model is unbounded')
    exit(0)
else:
    print('Optimization ended with status %d' % model.status)
    exit(0)

# Iterate over the solutions and compute the objectives
model.Params.outputFlag = 0
print('')
```

```
for k in range(model.solCount):
    model.Params.solutionNumber = k
    objn = 0
    for v in model.getVars():
        objn += v.obj * v.xn
    print('Solution %d has objective %g' % (k, objn))
print('')
model.Params.outputFlag = 1

fixed = model.fixed()
fixed.Params.presolve = 0
fixed.optimize()

if fixed.status != GRB.Status.OPTIMAL:
    print("Error: fixed model isn't optimal")
    exit(1)

diff = model.objVal - fixed.objVal

if abs(diff) > 1e-6 * (1.0 + abs(model.objVal)):
    print('Error: objective values are different')
    exit(1)

# Print values of nonzero variables
for v in fixed.getVars():
    if v.x != 0:
        print('%s %g' % (v.varName, v.x))
```

multiobj.py

```
#!/usr/bin/python

# Copyright 2017, Gurobi Optimization, Inc.

# Want to cover three different sets but subject to a common budget of
# elements allowed to be used. However, the sets have different priorities to
# be covered; and we tackle this by using multi-objective optimization.

from __future__ import print_function
from gurobipy import *

try:
    # Sample data
    Groundset = range(20)
    Subsets   = range(4)
    Budget    = 12;
    Set = [ [ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
            [ 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1 ],
            [ 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0 ],
            [ 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0 ] ]
    SetObjPriority = [  3,    2,    2,   1]
    SetObjWeight   = [1.0, 0.25, 1.25, 1.0]

    # Create initial model
    model = Model('multiobj')

    # Initialize decision variables for ground set:
    # x[e] == 1 if element e is chosen for the covering.
    Elem = model.addVars(Groundset, vtype=GRB.BINARY, name='El')

    # Constraint: limit total number of elements to be picked to be at most
    # Budget
    model.addConstr(Elem.sum() <= Budget, name='Budget')

    # Set global sense for ALL objectives
    model.ModelSense = GRB.MAXIMIZE

    # Limit how many solutions to collect
    model.setParam(GRB.Param.PoolSolutions, 100)

    # Set and configure i-th objective
    for i in Subsets:
        objn = sum(Elem[k]*Set[i][k] for k in range(len(Elem)))
        model.setObjectiveN(objn, i, SetObjPriority[i], SetObjWeight[i],
```

```
                        1.0 + i, 0.01, 'Set' + str(i))

# Save problem
model.write('multiobj.lp')

# Optimize
model.optimize()

model.setParam(GRB.Param.OutputFlag, 0)

# Status checking
status = model.Status
if status == GRB.Status.INF_OR_UNBD or \
   status == GRB.Status.INFEASIBLE  or \
   status == GRB.Status.UNBOUNDED:
    print('The model cannot be solved because it is infeasible or unbounded')
    sys.exit(1)

if status != GRB.Status.OPTIMAL:
    print('Optimization was stopped with status ' + str(status))
    sys.exit(1)

# Print best selected set
print('Selected elements in best solution:')
for e in Groundset:
    if Elem[e].X > 0.9:
        print(' El%d' % e, end='')
print('')

# Print number of solutions stored
nSolutions = model.SolCount
print('Number of solutions found: ' + str(nSolutions))

# Print objective values of solutions
if nSolutions > 10:
    nSolutions = 10
print('Objective values for first ' + str(nSolutions) + ' solutions:')
for i in Subsets:
    model.setParam(GRB.Param.ObjNumber, i)
    print('\tSet%d' % i, end='')
    for e in range(nSolutions):
        model.setParam(GRB.Param.SolutionNumber, e)
        print(' %6g' % model.ObjNVal, end='')
    print('')
```

```
except GurobiError as e:
    print('Error code ' + str(e.errno) + ": " + str(e))

except AttributeError as e:
    print('Encountered an attribute error: ' + str(e))
```

netflow.py

```
#!/usr/bin/python

# Copyright 2017, Gurobi Optimization, Inc.

# Solve a multi-commodity flow problem.  Two products ('Pencils' and 'Pens')
# are produced in 2 cities ('Detroit' and 'Denver') and must be sent to
# warehouses in 3 cities ('Boston', 'New York', and 'Seattle') to
# satisfy demand ('inflow[h,i]').
#
# Flows on the transportation network must respect arc capacity constraints
# ('capacity[i,j]'). The objective is to minimize the sum of the arc
# transportation costs ('cost[i,j]').

from gurobipy import *

# Model data

commodities = ['Pencils', 'Pens']
nodes = ['Detroit', 'Denver', 'Boston', 'New York', 'Seattle']

arcs, capacity = multidict({
  ('Detroit', 'Boston'):   100,
  ('Detroit', 'New York'):  80,
  ('Detroit', 'Seattle'):  120,
  ('Denver',  'Boston'):   120,
  ('Denver',  'New York'): 120,
  ('Denver',  'Seattle'):  120 })

cost = {
  ('Pencils', 'Detroit', 'Boston'):   10,
  ('Pencils', 'Detroit', 'New York'): 20,
  ('Pencils', 'Detroit', 'Seattle'):  60,
  ('Pencils', 'Denver',  'Boston'):   40,
  ('Pencils', 'Denver',  'New York'): 40,
  ('Pencils', 'Denver',  'Seattle'):  30,
  ('Pens',    'Detroit', 'Boston'):   20,
  ('Pens',    'Detroit', 'New York'): 20,
  ('Pens',    'Detroit', 'Seattle'):  80,
  ('Pens',    'Denver',  'Boston'):   60,
  ('Pens',    'Denver',  'New York'): 70,
  ('Pens',    'Denver',  'Seattle'):  30 }

inflow = {
  ('Pencils', 'Detroit'):   50,
```

```
  ('Pencils', 'Denver'):    60,
  ('Pencils', 'Boston'):   -50,
  ('Pencils', 'New York'): -50,
  ('Pencils', 'Seattle'):  -10,
  ('Pens',    'Detroit'):   60,
  ('Pens',    'Denver'):    40,
  ('Pens',    'Boston'):   -40,
  ('Pens',    'New York'): -30,
  ('Pens',    'Seattle'):  -30 }

# Create optimization model
m = Model('netflow')

# Create variables
flow = m.addVars(commodities, arcs, obj=cost, name="flow")

# Arc capacity constraints
m.addConstrs(
    (flow.sum('*',i,j) <= capacity[i,j] for i,j in arcs), "cap")

# Equivalent version using Python looping
# for i,j in arcs:
#   m.addConstr(sum(flow[h,i,j] for h in commodities) <= capacity[i,j],
#               "cap[%s,%s]" % (i, j))


# Flow conservation constraints
m.addConstrs(
    (flow.sum(h,'*',j) + inflow[h,j] == flow.sum(h,j,'*')
    for h in commodities for j in nodes), "node")
# Alternate version:
# m.addConstrs(
#   (quicksum(flow[h,i,j] for i,j in arcs.select('*',j)) + inflow[h,j] ==
#     quicksum(flow[h,j,k] for j,k in arcs.select(j,'*'))
#     for h in commodities for j in nodes), "node")

# Compute optimal solution
m.optimize()

# Print solution
if m.status == GRB.Status.OPTIMAL:
    solution = m.getAttr('x', flow)
    for h in commodities:
        print('\nOptimal flows for %s:' % h)
        for i,j in arcs:
```

```
if solution[h,i,j] > 0:
    print('%s -> %s: %g' % (i, j, solution[h,i,j]))
```

params.py

```
#!/usr/bin/python

# Copyright 2017, Gurobi Optimization, Inc.

# Use parameters that are associated with a model.
#
# A MIP is solved for a few seconds with different sets of parameters.
# The one with the smallest MIP gap is selected, and the optimization
# is resumed until the optimal solution is found.

import sys
from gurobipy import *

if len(sys.argv) < 2:
    print('Usage: params.py filename')
    quit()


# Read model and verify that it is a MIP
m = read(sys.argv[1])
if m.isMIP == 0:
    print('The model is not an integer program')
    exit(1)

# Set a 2 second time limit
m.Params.timeLimit = 2

# Now solve the model with different values of MIPFocus
bestModel = m.copy()
bestModel.optimize()
for i in range(1, 4):
    m.reset()
    m.Params.MIPFocus = i
    m.optimize()
    if bestModel.MIPGap > m.MIPGap:
        bestModel, m = m, bestModel # swap models

# Finally, delete the extra model, reset the time limit and
# continue to solve the best model to optimality
del m
bestModel.Params.timeLimit = "default"
bestModel.optimize()
print('Solved with MIPFocus: %d' % bestModel.Params.MIPFocus)
```

piecewise.py

```
#!/usr/bin/python

# Copyright 2017, Gurobi Optimization, Inc.

# This example considers the following separable, convex problem:
#
#   minimize    f(x) - y + g(z)
#   subject to  x + 2 y + 3 z <= 4
#               x +   y       >= 1
#               x,    y,    z <= 1
#
# where f(u) = exp(-u) and g(u) = 2 u^2 - 4 u, for all real u. It
# formulates and solves a simpler LP model by approximating f and
# g with piecewise-linear functions. Then it transforms the model
# into a MIP by negating the approximation for f, which corresponds
# to a non-convex piecewise-linear function, and solves it again.

from gurobipy import *
from math import exp

def f(u):
    return exp(-u)

def g(u):
    return 2 * u * u - 4 * u

try:

    # Create a new model

    m = Model()

    # Create variables

    lb = 0.0
    ub = 1.0

    x = m.addVar(lb, ub, name='x')
    y = m.addVar(lb, ub, name='y')
    z = m.addVar(lb, ub, name='z')

    # Set objective for y

    m.setObjective(-y)
```

```
# Add piecewise-linear objective functions for x and z

npts = 101
ptu = []
ptf = []
ptg = []

for i in range(npts):
    ptu.append(lb + (ub - lb) * i / (npts - 1))
    ptf.append(f(ptu[i]))
    ptg.append(g(ptu[i]))

m.setPWLObj(x, ptu, ptf)
m.setPWLObj(z, ptu, ptg)

# Add constraint: x + 2 y + 3 z <= 4

m.addConstr(x + 2 * y + 3 * z <= 4, 'c0')

# Add constraint: x + y >= 1

m.addConstr(x + y >= 1, 'c1')

# Optimize model as an LP

m.optimize()

print('IsMIP: %d' % m.IsMIP)
for v in m.getVars():
    print('%s %g' % (v.VarName, v.X))
print('Obj: %g' % m.ObjVal)
print('')

# Negate piecewise-linear objective function for x

for i in range(npts):
    ptf[i] = -ptf[i]

m.setPWLObj(x, ptu, ptf)

# Optimize model as a MIP

m.optimize()
```

```
    print('IsMIP: %d' % m.IsMIP)
    for v in m.getVars():
        print('%s %g' % (v.VarName, v.X))
    print('Obj: %g' % m.ObjVal)

except GurobiError as e:
    print('Error code ' + str(e.errno) + ": " + str(e))

except AttributeError:
    print('Encountered an attribute error')
```

poolsearch.py

```
#!/usr/bin/python

# Copyright 2017, Gurobi Optimization, Inc.

# We find alternative epsilon-optimal solutions to a given knapsack
# problem by using PoolSearchMode

from __future__ import print_function
from gurobipy import *

try:
    # Sample data
    Groundset    = range(10)
    objCoef      = [32, 32, 15, 15, 6, 6, 1, 1, 1, 1]
    knapsackCoef = [16, 16,  8,  8, 4, 4, 2, 2, 1, 1]
    Budget       = 33

    # Create initial model
    model = Model("poolsearch")

    # Create dicts for tupledict.prod() function
    objCoefDict = dict(zip(Groundset, objCoef))
    knapsackCoefDict = dict(zip(Groundset, knapsackCoef))

    # Initialize decision variables for ground set:
    # x[e] == 1 if element e is chosen
    Elem = model.addVars(Groundset, vtype=GRB.BINARY, name='El')

    # Set objective function
    model.ModelSense = GRB.MAXIMIZE
    model.setObjective(Elem.prod(objCoefDict))

    # Constraint: limit total number of elements to be picked to be at most
    # Budget
    model.addConstr(Elem.prod(knapsackCoefDict) <= Budget, name='Budget')

    # Limit how many solutions to collect
    model.setParam(GRB.Param.PoolSolutions, 1024)
    # Limit the search space by setting a gap for the worst possible solution that will be accep
    model.setParam(GRB.Param.PoolGap, 0.10)
    # do a systematic search for the k-best solutions
    model.setParam(GRB.Param.PoolSearchMode, 2)

    # save problem
```

```
model.write('poolsearch.lp')

# Optimize
model.optimize()

model.setParam(GRB.Param.OutputFlag, 0)

# Status checking
status = model.Status
if status == GRB.Status.INF_OR_UNBD or \
   status == GRB.Status.INFEASIBLE  or \
   status == GRB.Status.UNBOUNDED:
    print('The model cannot be solved because it is infeasible or unbounded')
    sys.exit(1)

if status != GRB.Status.OPTIMAL:
    print('Optimization was stopped with status ' + str(status))
    sys.exit(1)

# Print best selected set
print('Selected elements in best solution:')
print('\t', end='')
for e in Groundset:
    if Elem[e].X > .9:
        print(' El%d' % e, end='')
print('')

# Print number of solutions stored
nSolutions = model.SolCount
print('Number of solutions found: ' + str(nSolutions))

# Print objective values of solutions
for e in range(nSolutions):
    model.setParam(GRB.Param.SolutionNumber, e)
    print('%g ' % model.PoolObjVal, end='')
    if e % 15 == 14:
        print('')
print('')

# print fourth best set if available
if (nSolutions >= 4):
    model.setParam(GRB.Param.SolutionNumber, 3);

    print('Selected elements in fourth best solution:')
    print('\t', end='')
```

```
        for e in Groundset:
            if Elem[e].Xn > .9:
                print(' El%d' % e, end='')
        print('')

except GurobiError as e:
    print('Gurobi error ' + str(e.errno) + ": " + str(e.message))

except AttributeError as e:
    print('Encountered an attribute error: ' + str(e))
```

portfolio.py

```
#!/usr/bin/python

# Copyright 2017, Gurobi Optimization, Inc.

# Portfolio selection: given a sum of money to invest, one must decide how to
# spend it amongst a portfolio of financial securities.  Our approach is due
# to Markowitz (1959) and looks to minimize the risk associated with the
# investment while realizing a target expected return.  By varying the target,
# one can compute an 'efficient frontier', which defines the optimal portfolio
# for a given expected return.
#
# Note that this example reads historical return data from a comma-separated
# file (../data/portfolio.csv).  As a result, it must be run from the Gurobi
# examples/python directory.
#
# This example requires the pandas, NumPy, and Matplotlib Python packages,
# which are part of the SciPy ecosystem for mathematics, science, and
# engineering (http://scipy.org).  These packages aren't included in all
# Python distributions, but are included by default with Anaconda Python.

from gurobipy import *
from math import sqrt
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Import (normalized) historical return data using pandas
data = pd.DataFrame.from_csv('../data/portfolio.csv')
stocks = data.columns

# Calculate basic summary statistics for individual stocks
stock_volatility = data.std()
stock_return = data.mean()

# Create an empty model
m = Model('portfolio')

# Add a variable for each stock
vars = pd.Series(m.addVars(stocks), index=stocks)

# Objective is to minimize risk (squared).  This is modeled using the
# covariance matrix, which measures the historical correlation between stocks.
sigma = data.cov()
portfolio_risk = sigma.dot(vars).dot(vars)
```

```
m.setObjective(portfolio_risk, GRB.MINIMIZE)

# Fix budget with a constraint
m.addConstr(vars.sum() == 1, 'budget')

# Optimize model to find the minimum risk portfolio
m.setParam('OutputFlag', 0)
m.optimize()

# Create an expression representing the expected return for the portfolio
portfolio_return = stock_return.dot(vars)

# Display minimum risk portfolio
print('Minimum Risk Portfolio:\n')
for v in vars:
    if v.x > 0:
        print('\t%s\t: %g' % (v.varname, v.x))
minrisk_volatility = sqrt(portfolio_risk.getValue())
print('\nVolatility      = %g' % minrisk_volatility)
minrisk_return = portfolio_return.getValue()
print('Expected Return = %g' % minrisk_return)

# Add (redundant) target return constraint
target = m.addConstr(portfolio_return == minrisk_return, 'target')

# Solve for efficient frontier by varying target return
frontier = pd.Series()
for r in np.linspace(stock_return.min(), stock_return.max(), 100):
    target.rhs = r
    m.optimize()
    frontier.loc[sqrt(portfolio_risk.getValue())] = r

# Plot volatility versus expected return for individual stocks
ax = plt.gca()
ax.scatter(x=stock_volatility, y=stock_return,
           color='Blue', label='Individual Stocks')
for i, stock in enumerate(stocks):
    ax.annotate(stock, (stock_volatility[i], stock_return[i]))

# Plot volatility versus expected return for minimum risk portfolio
ax.scatter(x=minrisk_volatility, y=minrisk_return, color='DarkGreen')
ax.annotate('Minimum\nRisk\nPortfolio', (minrisk_volatility, minrisk_return),
            horizontalalignment='right')

# Plot efficient frontier
```

```
frontier.plot(color='DarkGreen', label='Efficient Frontier', ax=ax)

# Format and display the final plot
ax.axis([0.005, 0.06, -0.02, 0.025])
ax.set_xlabel('Volatility (standard deviation)')
ax.set_ylabel('Expected Return')
ax.legend()
ax.grid()
plt.show()
```

qcp.py

```
#!/usr/bin/python

# Copyright 2017, Gurobi Optimization, Inc.

# This example formulates and solves the following simple QCP model:
#     maximize    x
#     subject to  x + y + z = 1
#                 x^2 + y^2 <= z^2 (second-order cone)
#                 x^2 <= yz        (rotated second-order cone)

from gurobipy import *

# Create a new model
m = Model("qcp")

# Create variables
x = m.addVar(name="x")
y = m.addVar(name="y")
z = m.addVar(name="z")

# Set objective: x
obj = 1.0*x
m.setObjective(obj, GRB.MAXIMIZE)

# Add constraint: x + y + z = 1
m.addConstr(x + y + z == 1, "c0")

# Add second-order cone: x^2 + y^2 <= z^2
m.addConstr(x*x + y*y <= z*z, "qc0")

# Add rotated cone: x^2 <= yz
m.addConstr(x*x <= y*z, "qc1")

m.optimize()

for v in m.getVars():
    print('%s %g' % (v.varName, v.x))

print('Obj: %g' % obj.getValue())
```

qp.py

```
#!/usr/bin/python

# Copyright 2017, Gurobi Optimization, Inc.

# This example formulates and solves the following simple QP model:
#  minimize
#      x^2 + x*y + y^2 + y*z + z^2 + 2 x
#  subject to
#      x + 2 y + 3 z >= 4
#      x +   y       >= 1
#
# It solves it once as a continuous model, and once as an integer model.

from gurobipy import *

# Create a new model
m = Model("qp")

# Create variables
x = m.addVar(ub=1.0, name="x")
y = m.addVar(ub=1.0, name="y")
z = m.addVar(ub=1.0, name="z")

# Set objective: x^2 + x*y + y^2 + y*z + z^2 + 2 x
obj = x*x + x*y + y*y + y*z + z*z + 2*x
m.setObjective(obj)

# Add constraint: x + 2 y + 3 z <= 4
m.addConstr(x + 2 * y + 3 * z >= 4, "c0")

# Add constraint: x + y >= 1
m.addConstr(x + y >= 1, "c1")

m.optimize()

for v in m.getVars():
    print('%s %g' % (v.varName, v.x))

print('Obj: %g' % obj.getValue())

x.vType = GRB.INTEGER
y.vType = GRB.INTEGER
z.vType = GRB.INTEGER
```

```
m.optimize()

for v in m.getVars():
    print('%s %g' % (v.varName, v.x))

print('Obj: %g' % obj.getValue())
```

sensitivity.py

```
#!/usr/bin/python

# Copyright 2017, Gurobi Optimization, Inc.

# A simple sensitivity analysis example which reads a MIP model
# from a file and solves it. Then each binary variable is set
# to 1-X, where X is its value in the optimal solution, and
# the impact on the objective function value is reported.

import sys
from gurobipy import *

if len(sys.argv) < 2:
    print('Usage: sensitivity.py filename')
    quit()

# Read and solve model

model = read(sys.argv[1])

if model.IsMIP == 0:
    print('Model is not a MIP')
    exit(0)

model.optimize()

if model.status != GRB.Status.OPTIMAL:
    print('Optimization ended with status %d' % model.status)
    exit(0)

# Store the optimal solution

origObjVal = model.ObjVal
for v in model.getVars():
    v._origX = v.X

# Disable solver output for subsequent solves

model.Params.outputFlag = 0

# Iterate through unfixed, binary variables in model

for v in model.getVars():
    if (v.LB == 0 and v.UB == 1 \
```

```
and (v.VType == GRB.BINARY or v.VType == GRB.INTEGER)):

# Set variable to 1-X, where X is its value in optimal solution

if v._origX < 0.5:
    v.LB = v.Start = 1
else:
    v.UB = v.Start = 0

# Update MIP start for the other variables

for vv in model.getVars():
    if not vv.sameAs(v):
        vv.Start = vv._origX

# Solve for new value and capture sensitivity information

model.optimize()

if model.status == GRB.Status.OPTIMAL:
    print('Objective sensitivity for variable %s is %g' % \
          (v.VarName, model.ObjVal - origObjVal))
else:
    print('Objective sensitivity for variable %s is infinite' % \
          v.VarName)

# Restore the original variable bounds

v.LB = 0
v.UB = 1
```

sos.py

```
#!/usr/bin/python

# Copyright 2017, Gurobi Optimization, Inc.

# This example creates a very simple Special Ordered Set (SOS) model.
# The model consists of 3 continuous variables, no linear constraints,
# and a pair of SOS constraints of type 1.

from gurobipy import *

try:

    # Create a new model

    model = Model("sos")

    # Create variables

    x0 = model.addVar(ub=1.0, name="x0")
    x1 = model.addVar(ub=1.0, name="x1")
    x2 = model.addVar(ub=2.0, name="x2")

    # Set objective
    model.setObjective(2 * x0 + x1 + x2, GRB.MAXIMIZE)

    # Add first SOS: x0 = 0 or x1 = 0
    model.addSOS(GRB.SOS_TYPE1, [x0, x1], [1, 2])

    # Add second SOS: x0 = 0 or x2 = 0
    model.addSOS(GRB.SOS_TYPE1, [x0, x2], [1, 2])

    model.optimize()

    for v in model.getVars():
        print('%s %g' % (v.varName, v.x))

    print('Obj: %g' % model.objVal)

except GurobiError as e:
    print('Error code ' + str(e.errno) + ": " + str(e))

except AttributeError:
    print('Encountered an attribute error')
```

sudoku.py

```
#!/usr/bin/python

# Copyright 2017, Gurobi Optimization, Inc.

# Sudoku example.

# The Sudoku board is a 9x9 grid, which is further divided into a 3x3 grid
# of 3x3 grids.  Each cell in the grid must take a value from 0 to 9.
# No two grid cells in the same row, column, or 3x3 subgrid may take the
# same value.
#
# In the MIP formulation, binary variables x[i,j,v] indicate whether
# cell <i,j> takes value 'v'.  The constraints are as follows:
#   1. Each cell must take exactly one value (sum_v x[i,j,v] = 1)
#   2. Each value is used exactly once per row (sum_i x[i,j,v] = 1)
#   3. Each value is used exactly once per column (sum_j x[i,j,v] = 1)
#   4. Each value is used exactly once per 3x3 subgrid (sum_grid x[i,j,v] = 1)
#
# Input datasets for this example can be found in examples/data/sudoku*.

import sys
import math
from gurobipy import *

if len(sys.argv) < 2:
    print('Usage: sudoku.py filename')
    quit()

f = open(sys.argv[1])

grid = f.read().split()

n = len(grid[0])
s = int(math.sqrt(n))


# Create our 3-D array of model variables

model = Model('sudoku')

vars = model.addVars(n,n,n, vtype=GRB.BINARY, name='G')


# Fix variables associated with cells whose values are pre-specified
```

```
for i in range(n):
    for j in range(n):
        if grid[i][j] != '.':
            v = int(grid[i][j]) - 1
            vars[i,j,v].LB = 1

# Each cell must take one value

model.addConstrs((vars.sum(i,j,'*') == 1
                  for i in range(n)
                  for j in range(n)), name='V')

# Each value appears once per row

model.addConstrs((vars.sum(i,'*',v) == 1
                  for i in range(n)
                  for v in range(n)), name='R')

# Each value appears once per column

model.addConstrs((vars.sum('*',j,v) == 1
                  for j in range(n)
                  for v in range(n)), name='C')


# Each value appears once per subgrid

model.addConstrs((
    quicksum(vars[i,j,v] for i in range(i0*s, (i0+1)*s)
                         for j in range(j0*s, (j0+1)*s)) == 1
              for v in range(n)
              for i0 in range(s)
              for j0 in range(s)), name='Sub')

model.optimize()

model.write('sudoku.lp')

print('')
print('Solution:')
print('')

# Retrieve optimization result
```

```
solution = model.getAttr('X', vars)

for i in range(n):
    sol = ''
    for j in range(n):
        for v in range(n):
            if solution[i,j,v] > 0.5:
                sol += str(v+1)
    print(sol)
```

tsp.py

```
#!/usr/bin/python

# Copyright 2017, Gurobi Optimization, Inc.

# Solve a traveling salesman problem on a randomly generated set of
# points using lazy constraints.   The base MIP model only includes
# 'degree-2' constraints, requiring each node to have exactly
# two incident edges.  Solutions to this model may contain subtours -
# tours that don't visit every city.  The lazy constraint callback
# adds new constraints to cut them off.

import sys
import math
import random
import itertools
from gurobipy import *

# Callback - use lazy constraints to eliminate sub-tours

def subtourelim(model, where):
    if where == GRB.Callback.MIPSOL:
        # make a list of edges selected in the solution
        vals = model.cbGetSolution(model._vars)
        selected = tuplelist((i,j) for i,j in model._vars.keys() if vals[i,j] > 0.5)
        # find the shortest cycle in the selected edge list
        tour = subtour(selected)
        if len(tour) < n:
            # add subtour elimination constraint for every pair of cities in tour
            model.cbLazy(quicksum(model._vars[i,j]
                                  for i,j in itertools.combinations(tour, 2))
                         <= len(tour)-1)


# Given a tuplelist of edges, find the shortest subtour

def subtour(edges):
    unvisited = list(range(n))
    cycle = range(n+1) # initial length has 1 more city
    while unvisited: # true if list is non-empty
        thiscycle = []
        neighbors = unvisited
        while neighbors:
            current = neighbors[0]
            thiscycle.append(current)
```

```
            unvisited.remove(current)
            neighbors = [j for i,j in edges.select(current,'*') if j in unvisited]
        if len(cycle) > len(thiscycle):
            cycle = thiscycle
    return cycle


# Parse argument

if len(sys.argv) < 2:
    print('Usage: tsp.py npoints')
    exit(1)
n = int(sys.argv[1])

# Create n random points

random.seed(1)
points = [(random.randint(0,100),random.randint(0,100)) for i in range(n)]

# Dictionary of Euclidean distance between each pair of points

dist = {(i,j) :
    math.sqrt(sum((points[i][k]-points[j][k])**2 for k in range(2)))
    for i in range(n) for j in range(i)}

m = Model()

# Create variables

vars = m.addVars(dist.keys(), obj=dist, vtype=GRB.BINARY, name='e')
for i,j in vars.keys():
    vars[j,i] = vars[i,j] # edge in opposite direction

# You could use Python looping constructs and m.addVar() to create
# these decision variables instead.  The following would be equivalent
# to the preceding m.addVars() call...
#
# vars = tupledict()
# for i,j in dist.keys():
#   vars[i,j] = m.addVar(obj=dist[i,j], vtype=GRB.BINARY,
#                        name='e[%d,%d]'%(i,j))


# Add degree-2 constraint
```

```
m.addConstrs(vars.sum(i,'*') == 2 for i in range(n))

# Using Python looping constructs, the preceding would be...
#
# for i in range(n):
#   m.addConstr(sum(vars[i,j] for j in range(n)) == 2)


# Optimize model

m._vars = vars
m.Params.lazyConstraints = 1
m.optimize(subtourelim)

vals = m.getAttr('x', vars)
selected = tuplelist((i,j) for i,j in vals.keys() if vals[i,j] > 0.5)

tour = subtour(selected)
assert len(tour) == n

print('')
print('Optimal tour: %s' % str(tour))
print('Optimal cost: %g' % m.objVal)
print('')
```

tune.py

```
#!/usr/bin/python

# Copyright 2017, Gurobi Optimization, Inc.

#  This example reads a model from a file and tunes it.
#  It then writes the best parameter settings to a file
#  and solves the model using these parameters.

import sys
from gurobipy import *

if len(sys.argv) < 2:
    print('Usage: tune.py filename')
    quit()

# Read the model
model = read(sys.argv[1])

# Set the TuneResults parameter to 1
model.Params.tuneResults = 1

# Tune the model
model.tune()

if model.tuneResultCount > 0:

    # Load the best tuned parameters into the model
    model.getTuneResult(0)

    # Write tuned parameters to a file
    model.write('tune.prm')

    # Solve the model using the tuned parameters
    model.optimize()
```

workforce1.py

```
#!/usr/bin/python

# Copyright 2017, Gurobi Optimization, Inc.

# Assign workers to shifts; each worker may or may not be available on a
# particular day. If the problem cannot be solved, use IIS to find a set of
# conflicting constraints. Note that there may be additional conflicts besides
# what is reported via IIS.

from gurobipy import *

# Number of workers required for each shift
shifts, shiftRequirements = multidict({
  "Mon1":  3,
  "Tue2":  2,
  "Wed3":  4,
  "Thu4":  4,
  "Fri5":  5,
  "Sat6":  6,
  "Sun7":  5,
  "Mon8":  2,
  "Tue9":  2,
  "Wed10": 3,
  "Thu11": 4,
  "Fri12": 6,
  "Sat13": 7,
  "Sun14": 5 })

# Amount each worker is paid to work one shift
workers, pay = multidict({
  "Amy":   10,
  "Bob":   12,
  "Cathy": 10,
  "Dan":   8,
  "Ed":    8,
  "Fred":  9,
  "Gu":    11 })

# Worker availability
availability = tuplelist([
('Amy', 'Tue2'), ('Amy', 'Wed3'), ('Amy', 'Fri5'), ('Amy', 'Sun7'),
('Amy', 'Tue9'), ('Amy', 'Wed10'), ('Amy', 'Thu11'), ('Amy', 'Fri12'),
('Amy', 'Sat13'), ('Amy', 'Sun14'), ('Bob', 'Mon1'), ('Bob', 'Tue2'),
('Bob', 'Fri5'), ('Bob', 'Sat6'), ('Bob', 'Mon8'), ('Bob', 'Thu11'),
```

```
('Bob', 'Sat13'), ('Cathy', 'Wed3'), ('Cathy', 'Thu4'), ('Cathy', 'Fri5'),
('Cathy', 'Sun7'), ('Cathy', 'Mon8'), ('Cathy', 'Tue9'), ('Cathy', 'Wed10'),
('Cathy', 'Thu11'), ('Cathy', 'Fri12'), ('Cathy', 'Sat13'),
('Cathy', 'Sun14'), ('Dan', 'Tue2'), ('Dan', 'Wed3'), ('Dan', 'Fri5'),
('Dan', 'Sat6'), ('Dan', 'Mon8'), ('Dan', 'Tue9'), ('Dan', 'Wed10'),
('Dan', 'Thu11'), ('Dan', 'Fri12'), ('Dan', 'Sat13'), ('Dan', 'Sun14'),
('Ed', 'Mon1'), ('Ed', 'Tue2'), ('Ed', 'Wed3'), ('Ed', 'Thu4'),
('Ed', 'Fri5'), ('Ed', 'Sun7'), ('Ed', 'Mon8'), ('Ed', 'Tue9'),
('Ed', 'Thu11'), ('Ed', 'Sat13'), ('Ed', 'Sun14'), ('Fred', 'Mon1'),
('Fred', 'Tue2'), ('Fred', 'Wed3'), ('Fred', 'Sat6'), ('Fred', 'Mon8'),
('Fred', 'Tue9'), ('Fred', 'Fri12'), ('Fred', 'Sat13'), ('Fred', 'Sun14'),
('Gu', 'Mon1'), ('Gu', 'Tue2'), ('Gu', 'Wed3'), ('Gu', 'Fri5'),
('Gu', 'Sat6'), ('Gu', 'Sun7'), ('Gu', 'Mon8'), ('Gu', 'Tue9'),
('Gu', 'Wed10'), ('Gu', 'Thu11'), ('Gu', 'Fri12'), ('Gu', 'Sat13'),
('Gu', 'Sun14')
])

# Model
m = Model("assignment")

# Assignment variables: x[w,s] == 1 if worker w is assigned to shift s.
# Since an assignment model always produces integer solutions, we use
# continuous variables and solve as an LP.
x = m.addVars(availability, ub=1, name="x")

# The objective is to minimize the total pay costs
m.setObjective(quicksum(pay[w]*x[w,s] for w,s in availability), GRB.MINIMIZE)

# Constraints: assign exactly shiftRequirements[s] workers to each shift s
reqCts = m.addConstrs((x.sum('*', s) == shiftRequirements[s]
                       for s in shifts), "_")

# Using Python looping constructs, the preceding statement would be...
#
# reqCts = {}
# for s in shifts:
#   reqCts[s] = m.addConstr(
#         quicksum(x[w,s] for w,s in availability.select('*', s)) ==
#         shiftRequirements[s], s)

# Optimize
m.optimize()
status = m.status
if status == GRB.Status.UNBOUNDED:
    print('The model cannot be solved because it is unbounded')
```

```
    exit(0)
if status == GRB.Status.OPTIMAL:
    print('The optimal objective is %g' % m.objVal)
    exit(0)
if status != GRB.Status.INF_OR_UNBD and status != GRB.Status.INFEASIBLE:
    print('Optimization was stopped with status %d' % status)
    exit(0)

# do IIS
print('The model is infeasible; computing IIS')
m.computeIIS()
print('\nThe following constraint(s) cannot be satisfied:')
for c in m.getConstrs():
    if c.IISConstr:
        print('%s' % c.constrName)
```

workforce2.py

```
#!/usr/bin/python

# Copyright 2017, Gurobi Optimization, Inc.

# Assign workers to shifts; each worker may or may not be available on a
# particular day. If the problem cannot be solved, use IIS iteratively to
# find all conflicting constraints.

from gurobipy import *

# Number of workers required for each shift
shifts, shiftRequirements = multidict({
  "Mon1":  3,
  "Tue2":  2,
  "Wed3":  4,
  "Thu4":  4,
  "Fri5":  5,
  "Sat6":  6,
  "Sun7":  5,
  "Mon8":  2,
  "Tue9":  2,
  "Wed10": 3,
  "Thu11": 4,
  "Fri12": 6,
  "Sat13": 7,
  "Sun14": 5 })

# Amount each worker is paid to work one shift
workers, pay = multidict({
  "Amy":   10,
  "Bob":   12,
  "Cathy": 10,
  "Dan":   8,
  "Ed":    8,
  "Fred":  9,
  "Gu":    11 })

# Worker availability
availability = tuplelist([
('Amy', 'Tue2'), ('Amy', 'Wed3'), ('Amy', 'Fri5'), ('Amy', 'Sun7'),
('Amy', 'Tue9'), ('Amy', 'Wed10'), ('Amy', 'Thu11'), ('Amy', 'Fri12'),
('Amy', 'Sat13'), ('Amy', 'Sun14'), ('Bob', 'Mon1'), ('Bob', 'Tue2'),
('Bob', 'Fri5'), ('Bob', 'Sat6'), ('Bob', 'Mon8'), ('Bob', 'Thu11'),
('Bob', 'Sat13'), ('Cathy', 'Wed3'), ('Cathy', 'Thu4'), ('Cathy', 'Fri5'),
```

```
('Cathy', 'Sun7'), ('Cathy', 'Mon8'), ('Cathy', 'Tue9'), ('Cathy', 'Wed10'),
('Cathy', 'Thu11'), ('Cathy', 'Fri12'), ('Cathy', 'Sat13'),
('Cathy', 'Sun14'), ('Dan', 'Tue2'), ('Dan', 'Wed3'), ('Dan', 'Fri5'),
('Dan', 'Sat6'), ('Dan', 'Mon8'), ('Dan', 'Tue9'), ('Dan', 'Wed10'),
('Dan', 'Thu11'), ('Dan', 'Fri12'), ('Dan', 'Sat13'), ('Dan', 'Sun14'),
('Ed', 'Mon1'), ('Ed', 'Tue2'), ('Ed', 'Wed3'), ('Ed', 'Thu4'),
('Ed', 'Fri5'), ('Ed', 'Sun7'), ('Ed', 'Mon8'), ('Ed', 'Tue9'),
('Ed', 'Thu11'), ('Ed', 'Sat13'), ('Ed', 'Sun14'), ('Fred', 'Mon1'),
('Fred', 'Tue2'), ('Fred', 'Wed3'), ('Fred', 'Sat6'), ('Fred', 'Mon8'),
('Fred', 'Tue9'), ('Fred', 'Fri12'), ('Fred', 'Sat13'), ('Fred', 'Sun14'),
('Gu', 'Mon1'), ('Gu', 'Tue2'), ('Gu', 'Wed3'), ('Gu', 'Fri5'),
('Gu', 'Sat6'), ('Gu', 'Sun7'), ('Gu', 'Mon8'), ('Gu', 'Tue9'),
('Gu', 'Wed10'), ('Gu', 'Thu11'), ('Gu', 'Fri12'), ('Gu', 'Sat13'),
('Gu', 'Sun14')
])

# Model
m = Model("assignment")

# Assignment variables: x[w,s] == 1 if worker w is assigned to shift s.
# Since an assignment model always produces integer solutions, we use
# continuous variables and solve as an LP.
x = m.addVars(availability, ub=1, name="x")

# The objective is to minimize the total pay costs
m.setObjective(quicksum(pay[w]*x[w,s] for w,s in availability), GRB.MINIMIZE)

# Constraint: assign exactly shiftRequirements[s] workers to each shift s
reqCts = m.addConstrs((x.sum('*', s) == shiftRequirements[s]
                       for s in shifts), "_")

# Optimize
m.optimize()
status = m.status
if status == GRB.Status.UNBOUNDED:
    print('The model cannot be solved because it is unbounded')
    exit(0)
if status == GRB.Status.OPTIMAL:
    print('The optimal objective is %g' % m.objVal)
    exit(0)
if status != GRB.Status.INF_OR_UNBD and status != GRB.Status.INFEASIBLE:
    print('Optimization was stopped with status %d' % status)
    exit(0)

# do IIS
```

```
print('The model is infeasible; computing IIS')
removed = []

# Loop until we reduce to a model that can be solved
while True:

    m.computeIIS()
    print('\nThe following constraint cannot be satisfied:')
    for c in m.getConstrs():
        if c.IISConstr:
            print('%s' % c.constrName)
            # Remove a single constraint from the model
            removed.append(str(c.constrName))
            m.remove(c)
            break
    print('')

    m.optimize()
    status = m.status

    if status == GRB.Status.UNBOUNDED:
        print('The model cannot be solved because it is unbounded')
        exit(0)
    if status == GRB.Status.OPTIMAL:
        break
    if status != GRB.Status.INF_OR_UNBD and status != GRB.Status.INFEASIBLE:
        print('Optimization was stopped with status %d' % status)
        exit(0)

print('\nThe following constraints were removed to get a feasible LP:')
print(removed)
```

workforce3.py

```
#!/usr/bin/python

# Copyright 2017, Gurobi Optimization, Inc.

# Assign workers to shifts; each worker may or may not be available on a
# particular day. If the problem cannot be solved, relax the model
# to determine which constraints cannot be satisfied, and how much
# they need to be relaxed.

from gurobipy import *

# Number of workers required for each shift
shifts, shiftRequirements = multidict({
  "Mon1":  3,
  "Tue2":  2,
  "Wed3":  4,
  "Thu4":  4,
  "Fri5":  5,
  "Sat6":  6,
  "Sun7":  5,
  "Mon8":  2,
  "Tue9":  2,
  "Wed10": 3,
  "Thu11": 4,
  "Fri12": 6,
  "Sat13": 7,
  "Sun14": 5 })

# Amount each worker is paid to work one shift
workers, pay = multidict({
  "Amy":   10,
  "Bob":   12,
  "Cathy": 10,
  "Dan":   8,
  "Ed":    8,
  "Fred":  9,
  "Gu":    11 })

# Worker availability
availability = tuplelist([
('Amy', 'Tue2'), ('Amy', 'Wed3'), ('Amy', 'Fri5'), ('Amy', 'Sun7'),
('Amy', 'Tue9'), ('Amy', 'Wed10'), ('Amy', 'Thu11'), ('Amy', 'Fri12'),
('Amy', 'Sat13'), ('Amy', 'Sun14'), ('Bob', 'Mon1'), ('Bob', 'Tue2'),
('Bob', 'Fri5'), ('Bob', 'Sat6'), ('Bob', 'Mon8'), ('Bob', 'Thu11'),
```

```
('Bob', 'Sat13'), ('Cathy', 'Wed3'), ('Cathy', 'Thu4'), ('Cathy', 'Fri5'),
('Cathy', 'Sun7'), ('Cathy', 'Mon8'), ('Cathy', 'Tue9'), ('Cathy', 'Wed10'),
('Cathy', 'Thu11'), ('Cathy', 'Fri12'), ('Cathy', 'Sat13'),
('Cathy', 'Sun14'), ('Dan', 'Tue2'), ('Dan', 'Wed3'), ('Dan', 'Fri5'),
('Dan', 'Sat6'), ('Dan', 'Mon8'), ('Dan', 'Tue9'), ('Dan', 'Wed10'),
('Dan', 'Thu11'), ('Dan', 'Fri12'), ('Dan', 'Sat13'), ('Dan', 'Sun14'),
('Ed', 'Mon1'), ('Ed', 'Tue2'), ('Ed', 'Wed3'), ('Ed', 'Thu4'),
('Ed', 'Fri5'), ('Ed', 'Sun7'), ('Ed', 'Mon8'), ('Ed', 'Tue9'),
('Ed', 'Thu11'), ('Ed', 'Sat13'), ('Ed', 'Sun14'), ('Fred', 'Mon1'),
('Fred', 'Tue2'), ('Fred', 'Wed3'), ('Fred', 'Sat6'), ('Fred', 'Mon8'),
('Fred', 'Tue9'), ('Fred', 'Fri12'), ('Fred', 'Sat13'), ('Fred', 'Sun14'),
('Gu', 'Mon1'), ('Gu', 'Tue2'), ('Gu', 'Wed3'), ('Gu', 'Fri5'),
('Gu', 'Sat6'), ('Gu', 'Sun7'), ('Gu', 'Mon8'), ('Gu', 'Tue9'),
('Gu', 'Wed10'), ('Gu', 'Thu11'), ('Gu', 'Fri12'), ('Gu', 'Sat13'),
('Gu', 'Sun14')
])

# Model
m = Model("assignment")

# Assignment variables: x[w,s] == 1 if worker w is assigned to shift s.
# Since an assignment model always produces integer solutions, we use
# continuous variables and solve as an LP.
x = m.addVars(availability, ub=1, name="x")

# The objective is to minimize the total pay costs
m.setObjective(quicksum(pay[w]*x[w,s] for w,s in availability), GRB.MINIMIZE)

# Constraint: assign exactly shiftRequirements[s] workers to each shift s
reqCts = m.addConstrs((x.sum('*', s) == shiftRequirements[s]
                       for s in shifts), "_")

# Optimize
m.optimize()
status = m.status
if status == GRB.Status.UNBOUNDED:
    print('The model cannot be solved because it is unbounded')
    exit(0)
if status == GRB.Status.OPTIMAL:
    print('The optimal objective is %g' % m.objVal)
    exit(0)
if status != GRB.Status.INF_OR_UNBD and status != GRB.Status.INFEASIBLE:
    print('Optimization was stopped with status %d' % status)
    exit(0)
```

```
# Relax the constraints to make the model feasible
print('The model is infeasible; relaxing the constraints')
orignumvars = m.NumVars
m.feasRelaxS(0, False, False, True)
m.optimize()
status = m.status
if status in (GRB.Status.INF_OR_UNBD, GRB.Status.INFEASIBLE, GRB.Status.UNBOUNDED):
    print('The relaxed model cannot be solved \
           because it is infeasible or unbounded')
    exit(1)

if status != GRB.Status.OPTIMAL:
    print('Optimization was stopped with status %d' % status)
    exit(1)

print('\nSlack values:')
slacks = m.getVars()[orignumvars:]
for sv in slacks:
    if sv.X > 1e-6:
        print('%s = %g' % (sv.VarName, sv.X))
```

workforce4.py

```
#!/usr/bin/python

# Copyright 2017, Gurobi Optimization, Inc.

# Assign workers to shifts; each worker may or may not be available on a
# particular day. We use lexicographic optimization to solve the model:
# first, we minimize the linear sum of the slacks. Then, we constrain
# the sum of the slacks, and we minimize a quadratic objective that
# tries to balance the workload among the workers.

from gurobipy import *

# Number of workers required for each shift
shifts, shiftRequirements = multidict({
  "Mon1":  3,
  "Tue2":  2,
  "Wed3":  4,
  "Thu4":  4,
  "Fri5":  5,
  "Sat6":  6,
  "Sun7":  5,
  "Mon8":  2,
  "Tue9":  2,
  "Wed10": 3,
  "Thu11": 4,
  "Fri12": 6,
  "Sat13": 7,
  "Sun14": 5 })

# Amount each worker is paid to work one shift
workers, pay = multidict({
  "Amy":   10,
  "Bob":   12,
  "Cathy": 10,
  "Dan":   8,
  "Ed":    8,
  "Fred":  9,
  "Gu":    11 })

# Worker availability
availability = tuplelist([
('Amy', 'Tue2'), ('Amy', 'Wed3'), ('Amy', 'Fri5'), ('Amy', 'Sun7'),
('Amy', 'Tue9'), ('Amy', 'Wed10'), ('Amy', 'Thu11'), ('Amy', 'Fri12'),
('Amy', 'Sat13'), ('Amy', 'Sun14'), ('Bob', 'Mon1'), ('Bob', 'Tue2'),
```

```
('Bob', 'Fri5'), ('Bob', 'Sat6'), ('Bob', 'Mon8'), ('Bob', 'Thu11'),
('Bob', 'Sat13'), ('Cathy', 'Wed3'), ('Cathy', 'Thu4'), ('Cathy', 'Fri5'),
('Cathy', 'Sun7'), ('Cathy', 'Mon8'), ('Cathy', 'Tue9'), ('Cathy', 'Wed10'),
('Cathy', 'Thu11'), ('Cathy', 'Fri12'), ('Cathy', 'Sat13'),
('Cathy', 'Sun14'), ('Dan', 'Tue2'), ('Dan', 'Wed3'), ('Dan', 'Fri5'),
('Dan', 'Sat6'), ('Dan', 'Mon8'), ('Dan', 'Tue9'), ('Dan', 'Wed10'),
('Dan', 'Thu11'), ('Dan', 'Fri12'), ('Dan', 'Sat13'), ('Dan', 'Sun14'),
('Ed', 'Mon1'), ('Ed', 'Tue2'), ('Ed', 'Wed3'), ('Ed', 'Thu4'),
('Ed', 'Fri5'), ('Ed', 'Sun7'), ('Ed', 'Mon8'), ('Ed', 'Tue9'),
('Ed', 'Thu11'), ('Ed', 'Sat13'), ('Ed', 'Sun14'), ('Fred', 'Mon1'),
('Fred', 'Tue2'), ('Fred', 'Wed3'), ('Fred', 'Sat6'), ('Fred', 'Mon8'),
('Fred', 'Tue9'), ('Fred', 'Fri12'), ('Fred', 'Sat13'), ('Fred', 'Sun14'),
('Gu', 'Mon1'), ('Gu', 'Tue2'), ('Gu', 'Wed3'), ('Gu', 'Fri5'),
('Gu', 'Sat6'), ('Gu', 'Sun7'), ('Gu', 'Mon8'), ('Gu', 'Tue9'),
('Gu', 'Wed10'), ('Gu', 'Thu11'), ('Gu', 'Fri12'), ('Gu', 'Sat13'),
('Gu', 'Sun14')
])

# Model
m = Model("assignment")

# Assignment variables: x[w,s] == 1 if worker w is assigned to shift s.
# This is no longer a pure assignment model, so we must use binary variables.
x = m.addVars(availability, vtype=GRB.BINARY, name="x")

# Slack variables for each shift constraint so that the shifts can
# be satisfied
slacks = m.addVars(shifts, name="Slack")

# Variable to represent the total slack
totSlack = m.addVar(name="totSlack")

# Variables to count the total shifts worked by each worker
totShifts = m.addVars(workers, name="TotShifts")

# Constraint: assign exactly shiftRequirements[s] workers to each shift s,
# plus the slack
reqCts = m.addConstrs((slacks[s] + x.sum('*', s) == shiftRequirements[s]
                       for s in shifts), "_")

# Constraint: set totSlack equal to the total slack
m.addConstr(totSlack == slacks.sum(), "totSlack")

# Constraint: compute the total number of shifts for each worker
m.addConstrs((totShifts[w] == x.sum(w) for w in workers), "totShifts")
```

```
# Objective: minimize the total slack
# Note that this replaces the previous 'pay' objective coefficients
m.setObjective(totSlack)


# Optimize
def solveAndPrint():
    m.optimize()
    status = m.status
    if status == GRB.Status.INF_OR_UNBD or status == GRB.Status.INFEASIBLE \
      or status == GRB.Status.UNBOUNDED:
        print('The model cannot be solved because it is infeasible or \
               unbounded')
        exit(1)

    if status != GRB.Status.OPTIMAL:
        print('Optimization was stopped with status %d' % status)
        exit(0)

    # Print total slack and the number of shifts worked for each worker
    print('')
    print('Total slack required: %g' % totSlack.x)
    for w in workers:
        print('%s worked %g shifts' % (w, totShifts[w].x))
    print('')

solveAndPrint()

# Constrain the slack by setting its upper and lower bounds
totSlack.ub = totSlack.x
totSlack.lb = totSlack.x

# Variable to count the average number of shifts worked
avgShifts = m.addVar(name="avgShifts")

# Variables to count the difference from average for each worker;
# note that these variables can take negative values.
diffShifts = m.addVars(workers, lb=-GRB.INFINITY, name="Diff")

# Constraint: compute the average number of shifts worked
m.addConstr(len(workers) * avgShifts == totShifts.sum(), "avgShifts")

# Constraint: compute the difference from the average number of shifts
m.addConstrs((diffShifts[w] == totShifts[w] - avgShifts for w in workers),
```

```
            "Diff")

# Objective: minimize the sum of the square of the difference from the
# average number of shifts worked
m.setObjective(quicksum(diffShifts[w]*diffShifts[w] for w in workers))

# Optimize
solveAndPrint()
```

workforce5.py

```
#!/usr/bin/python

# Copyright 2017, Gurobi Optimization, Inc.

# Assign workers to shifts; each worker may or may not be available on a
# particular day. We use multi-objective optimization to solve the model.
# The highest-priority objective minimizes the sum of the slacks
# (i.e., the total number of uncovered shifts). The secondary objective
# minimizes the difference between the maximum and minimum number of
# shifts worked among all workers.  The second optimization is allowed
# to degrade the first objective by up to the smaller value of 10% and 2 */

from gurobipy import *

try:
    # Sample data
    # Sets of days and workers
    Shifts = [ "Mon1", "Tue2", "Wed3", "Thu4", "Fri5", "Sat6",
               "Sun7", "Mon8", "Tue9", "Wed10", "Thu11", "Fri12", "Sat13",
               "Sun14" ]
    Workers = [ "Amy", "Bob", "Cathy", "Dan", "Ed", "Fred", "Gu", "Tobi" ]

    # Number of workers required for each shift
    S = [ 3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5 ]
    shiftRequirements = { s : S[i] for i,s in enumerate(Shifts) }

    # Worker availability: 0 if the worker is unavailable for a shift
    A = [ [ 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1 ],
          [ 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0 ],
          [ 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 ],
          [ 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1 ],
          [ 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1 ],
          [ 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1 ],
          [ 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1 ],
          [ 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 ] ]
    availability = { (w,s) : A[j][i] for i,s in enumerate(Shifts)
                                     for j,w in enumerate(Workers) }

    # Create initial model
    model = Model("workforce5")

    # Initialize assignment decision variables:
    # x[w][s] == 1 if worker w is assigned to shift s.
    # This is no longer a pure assignment model, so we must
```

```
# use binary variables.
x = model.addVars(availability.keys(), ub=availability, vtype=GRB.BINARY,
                  name='x')

# Slack variables for each shift constraint so that the shifts can
# be satisfied
slacks = model.addVars(Shifts, name='Slack')

# Variable to represent the total slack
totSlack = model.addVar(name='totSlack')

# Variables to count the total shifts worked by each worker
totShifts = model.addVars(Workers, name='TotShifts')

# Constraint: assign exactly shiftRequirements[s] workers
# to each shift s, plus the slack
model.addConstrs((x.sum('*',s) + slacks[s] == shiftRequirements[s] for s in Shifts),
                 name='shiftRequirement')

# Constraint: set totSlack equal to the total slack
model.addConstr(totSlack == slacks.sum(), name='totSlack')

# Constraint: compute the total number of shifts for each worker
model.addConstrs((totShifts[w] == x.sum(w,'*') for w in Workers),
                 name='totShifts')

# Constraint: set minShift/maxShift variable to less/greater than the
# number of shifts among all workers
minShift = model.addVar(name='minShift')
maxShift = model.addVar(name='maxShift')
model.addGenConstrMin(minShift, totShifts, name='minShift')
model.addGenConstrMax(maxShift, totShifts, name='maxShift')

# Set global sense for ALL objectives
model.ModelSense = GRB.MINIMIZE

# Set up primary objective
model.setObjectiveN(totSlack, index=0, priority=2, abstol=2.0, reltol=0.1,
                    name='TotalSlack')

# Set up secondary objective
model.setObjectiveN(maxShift - minShift, index=1, priority=1,
                    name='Fairness')

# Save problem
```

```
    model.write('workforce5.lp')

    # Optimize
    model.optimize()

    status = model.Status
    if status == GRB.Status.INF_OR_UNBD or \
       status == GRB.Status.INFEASIBLE  or \
       status == GRB.Status.UNBOUNDED:
        print('The model cannot be solved because it is infeasible or unbounded')
        sys.exit(0)

    if status != GRB.Status.OPTIMAL:
        print('Optimization was stopped with status ' + str(status))
        sys.exit(0)

    # Print total slack and the number of shifts worked for each worker
    print('')
    print('Total slack required: ' + str(totSlack.X))
    for w in Workers:
      print(w + ' worked ' + str(totShifts[w].X) + ' shifts')
    print('')

except GurobiError as e:
    print('Error code ' + str(e.errno) + ": " + str(e))

except AttributeError as e:
    print('Encountered an attribute error: ' + str(e))
```

3.7 MATLAB Examples

This section includes source code for all of the Gurobi MATLAB examples. The same source code can be found in the `examples/matlab` directory of the Gurobi distribution.

diet.m

```
function diet()
% diet Solve the classic diet model

% Copyright 2017, Gurobi Optimization, Inc


% Nutrition guidelines, based on
% USDA Dietary Guidelines for Americans, 2005
% http://www.health.gov/DietaryGuidelines/dga2005/

ncategories = 4;
categories = {'calories'; 'protein'; 'fat'; 'sodium'};
%             minNutrition maxNutrition
categorynutrition = [ 1800 2200;   % calories
                        91   inf;    % protein
                        0    65;     % fat
                        0    1779];  % sodium

nfoods = 9;
foods = {'hamburger';
         'chicken';
         'hot dog';
         'fries';
         'macaroni';
         'pizza';
         'salad';
         'milk';
         'ice cream'};

foodcost = [2.49;  % hamburger
            2.89;  % chicken
            1.50;  % hot dog
            1.89;  % fries
            2.09;  % macaroni
            1.99;  % pizza
            2.49;  % salad
            0.89;  % milk
            1.59]; % ice cream
```

```
                  % calories protein fat sodium
nutritionValues = [ 410      24      26  730;   % hamburger
                    420      32      10  1190;  % chicken
                    560      20      32  1800;  % hot dog
                    380      4       19  270;   % fries
                    320      12      10  930;   % macaroni
                    320      15      12  820;   % pizza
                    320      31      12  1230;  % salad
                    100      8       2.5 125;   % milk
                    330      8       10  180];  % ice cream
nutritionValues = sparse(nutritionValues);
model.modelName = 'diet';

% The variables are layed out as [ buy; nutrition]
model.obj   = [ foodcost;         zeros(ncategories, 1)];
model.lb    = [ zeros(nfoods, 1); categorynutrition(:, 1)];
model.ub    = [  inf(nfoods, 1);  categorynutrition(:, 2)];
model.A     = [ nutritionValues' -speye(ncategories)];
model.rhs   = zeros(ncategories, 1);
model.sense = repmat('=', ncategories, 1);


function printSolution(result)
    if strcmp(result.status, 'OPTIMAL')
        buy       = result.x(1:nfoods);
        nutrition = result.x(nfoods+1:nfoods+ncategories);
        fprintf('\nCost: %f\n', result.objval);
        fprintf('\nBuy:\n')
        for f=1:nfoods
            if buy(f) > 0.0001
                fprintf('%10s %g\n', foods{f}, buy(f));
            end
        end
        fprintf('\nNutrition:\n')
        for c=1:ncategories
            fprintf('%10s %g\n', categories{c}, nutrition(c));
        end
    else
        fprintf('No solution\n');
    end
end

% Solve
results = gurobi(model);
printSolution(results);
```

```
fprintf('\nAdding constraint at most 6 servings of dairy\n')
milk = find(strcmp('milk', foods));
icecream = find(strcmp('ice cream', foods));
model.A(end+1,:) = sparse([1; 1], [milk; icecream], 1, ...
        1, nfoods + ncategories);
model.rhs(end+1) = 6;
model.sense(end+1) = '<';

% Solve
results = gurobi(model);
printSolution(results)

end
```

intlinprog.m

```
function [x, fval, exitflag] = intlinprog(f, intcon, A, b, Aeq, beq, lb, ub)
%INTLINPROG A mixed integer linear programming example using the
%   Gurobi MATLAB interface
%
%   This example is based on the intlinprog interface defined in the
%   MATLAB Optimization Toolbox. The Optimization Toolbox
%   is a registered trademark of The MathWorks, Inc.
%
%   x = INTLINPROG(f,intcon,A,b) solves the problem:
%
%   minimize     f'*x
%   subject to   A*x <= b
%                x(j) integer, when j is in the vector
%                intcon of integer constraints
%
%   x = INTLINPROG(f,intcon,A,b,Aeq,beq) solves the problem:
%
%   minimize     f'*x
%   subject to     A*x <= b,
%                Aeq*x == beq
%                x(j) integer, where j is in the vector
%                intcon of integer constraints
%
%   x = INTLINPROG(f,intcon,A,b,Aeq,beq,lb,ub) solves the problem:
%
%   minimize     f'*x
%   subject to     A*x <= b,
%                Aeq*x == beq,
%          lb <=     x <= ub.
%                x(j) integer, where j is in the vector
%                intcon of integer constraints
%
%   You can set lb(j) = -inf, if x(j) has no lower bound,
%   and ub(j) = inf, if x(j) has no upper bound.
%
%   [x, fval] = INTLINPROG(f, intcon, A, b) returns the objective value
%   at the solution. That is, fval = f'*x.
%
%   [x, fval, exitflag] = INTLINPROG(f, intcon, A, b) returns an exitflag
%   containing the status of the optimization. The values for
%   exitflag and corresponding status codes are:
%    2 - Solver stopped prematurely. Integer feasible point found.
%    1 - Optimal solution found.
%    0 - Solver stopped prematurely. No integer feasible point found.
```

```
%   -2 - No feasible point found.
%   -3 - Problem is unbounded.

if nargin < 4
    error('intlinprog(f, intcon, A, b)')
end

if nargin > 8
    error('intlinprog(f, intcon, A, b, Aeq, beq, lb, ub)');
end

if ~isempty(A)
    n = size(A, 2);
elseif nargin > 5 && ~isempty(Aeq)
    n = size(Aeq, 2);
else
    error('No linear constraints specified')
end

if ~issparse(A)
    A = sparse(A);
end

if nargin > 4 && ~issparse(Aeq)
    Aeq = sparse(Aeq);
end

model.obj = f;
model.vtype = repmat('C', n, 1);
model.vtype(intcon) = 'I';

if nargin < 5
    model.A = A;
    model.rhs = b;
    model.sense = '<';
else
    model.A = [A; Aeq];
    model.rhs = [b; beq];
    model.sense = [repmat('<', size(A,1), 1); repmat('=', size(Aeq,1), 1)];
end

if nargin < 7
    model.lb = -inf(n,1);
else
    model.lb = lb;
```

```
end

if nargin == 8
   model.ub = ub;
end

params.outputflag = 1;
result = gurobi(model, params);


if strcmp(result.status, 'OPTIMAL')
    exitflag = 1;
elseif strcmp(result.status, 'INTERRUPTED')
    if isfield(result, 'x')
        exitflag = 2;
    else
        exitflag = 0;
    end
elseif strcmp(result.status, 'INF_OR_UNBD')
    params.dualreductions = 0;
    result = gurobi(model, params);
    if strcmp(result.status, 'INFEASIBLE')
        exitflag = -2;
    elseif strcmp(result.status, 'UNBOUNDED')
        exitflag = -3;
    else
        exitflag = nan;
    end
else
    exitflag = nan;
end


if isfield(result, 'x')
    x = result.x;
else
    x = nan(n,1);
end

if isfield(result, 'objval')
    fval = result.objval;
else
    fval = nan;
end
```

linprog.m

```
function [x, fval, exitflag] = linprog(f, A, b, Aeq, beq, lb, ub)
%LINPROG A linear programming example using the Gurobi MATLAB interface
%
%   This example is based on the linprog interface defined in the
%   MATLAB Optimization Toolbox. The Optimization Toolbox
%   is a registered trademark of The MathWorks, Inc.
%
%   x = LINPROG(f,A,b) solves the linear programming problem:
%
%   minimize     f'*x
%   subject to   A*x <= b
%
%
%   x = LINPROG(f,A,b,Aeq,beq) solves the problem:
%
%   minimize     f'*x
%   subject to     A*x <= b,
%                Aeq*x == beq.
%
%   x = LINPROG(f,A,b,Aeq,beq,lb,ub) solves the problem:
%
%   minimize     f'*x
%   subject to     A*x <= b,
%                Aeq*x == beq,
%          lb <=     x <= ub.
%
%   You can set lb(j) = -inf, if x(j) has no lower bound,
%   and ub(j) = inf, if x(j) has no upper bound.
%
%   [x, fval] = LINPROG(f, A, b) returns the objective value
%   at the solution. That is, fval = f'*x.
%
%   [x, fval, exitflag] = LINPROG(f, A, b) returns an exitflag
%   containing the status of the optimization. The values for
%   exitflag and corresponding status codes are:
%      1 - OPTIMAL,
%      0 - ITERATION_LIMIT,
%     -2 - INFEASIBLE,
%     -3 - UNBOUNDED.
%

if nargin < 3
    error('linprog(f, A, b)')
end
```

```
if nargin > 7
    error('linprog(f, A, b, Aeq, beq, lb, ub)');
end

if ~isempty(A)
    n = size(A, 2);
elseif nargin > 4 && ~isempty(Aeq)
    n = size(Aeq, 2);
else
    error('No linear constraints specified')
end

if ~issparse(A)
    A = sparse(A);
end

if nargin > 3 && ~issparse(Aeq)
    Aeq = sparse(Aeq);
end


model.obj = f;

if nargin < 4
    model.A = A;
    model.rhs = b;
    model.sense = '<';
else
    model.A = [A; Aeq];
    model.rhs = [b; beq];
    model.sense = [repmat('<', size(A,1), 1); repmat('=', size(Aeq,1), 1)];
end

if nargin < 6
    model.lb = -inf(n,1);
else
    model.lb = lb;
end

if nargin == 7
   model.ub = ub;
end

params.outputflag = 0;
```

```
result = gurobi(model, params);


if strcmp(result.status, 'OPTIMAL')
    exitflag = 1;
elseif strcmp(result.status, 'ITERATION_LIMIT')
    exitflag = 0;
elseif strcmp(result.status, 'INF_OR_UNBD')
    params.dualreductions = 0;
    result = gurobi(model, params);
    if strcmp(result.status, 'INFEASIBLE')
        exitflag = -2;
    elseif strcmp(result.status, 'UNBOUNDED')
        exitflag = -3;
    else
        exitflag = nan;
    end
elseif strcmp(result.status, 'INFEASIBLE')
    exitflag = -2;
elseif strcmp(result.status, 'UNBOUNDED')
    exitflag = -3;
else
    exitflag = nan;
end


if isfield(result, 'x')
    x = result.x;
else
    x = nan(n,1);
end

if isfield(result, 'objval')
    fval = result.objval;
else
    fval = nan;
end
```

lp.m

```
% Copyright 2017, Gurobi Optimization, Inc.
%
% This example formulates and solves the following simple LP model:
% maximize
%       x + 2 y + 3 z
% subject to
%       x +   y        <= 1
%             y +   z  <= 1
%
clear model;
model.A = sparse([1 1 0; 0 1 1]);
model.obj = [1 2 3];
model.modelsense = 'Max';
model.rhs = [1 1];
model.sense = [ '<' '<'];

result = gurobi(model)

disp(result.objval);
disp(result.x);

% Alterantive representation of A - as sparse triplet matrix
i = [1; 1; 2; 2];
j = [1; 2; 2; 3];
x = [1; 1; 1; 1];
model.A = sparse(i, j, x, 2, 3);

clear params;
params.method = 2;
params.timelimit = 100;

result = gurobi(model, params);

disp(result.objval);
disp(result.x)
```

lp2.m

```
% Copyright 2017, Gurobi Optimization, Inc.
%
% Formulate a simple linear program, solve it, and then solve it
% again using the optimal basis.

clear model;
model.A = sparse([1 3 4; 8 2 3]);
model.obj = [1 2 3];
model.rhs = [4 7];
model.sense = ['>' '>'];

% First solve requires a few simplex iterations
result = gurobi(model)

model.vbasis = result.vbasis;
model.cbasis = result.cbasis;

% Second solve - start from an optimal basis, so no iterations

result = gurobi(model)
```

mip1.m

```
% Copyright 2017, Gurobi Optimization, Inc.

% This example formulates and solves the following simple MIP model:
%  maximize
%        x +   y + 2 z
%  subject to
%        x + 2 y + 3 z <= 4
%        x +   y       >= 1
%  x, y, z binary

names = {'x'; 'y'; 'z'};

try
    clear model;
    model.A = sparse([1 2 3; 1 1 0]);
    model.obj = [1 1 2];
    model.rhs = [4; 1];
    model.sense = '<>';
    model.vtype = 'B';
    model.modelsense = 'max';
    model.varnames = names;

    gurobi_write(model, 'mip1.lp');

    clear params;
    params.outputflag = 0;

    result = gurobi(model, params);

    disp(result)

    for v=1:length(names)
        fprintf('%s %d\n', names{v}, result.x(v));
    end

    fprintf('Obj: %e\n', result.objval);

catch gurobiError
    fprintf('Error reported\n');
end
```

piecewise.m

```
% Copyright 2017, Gurobi Optimization, Inc.

% This example considers the following separable, convex problem:
%
%   minimize    f(x) - y + g(z)
%   subject to  x + 2 y + 3 z <= 4
%               x +   y       >= 1
%               x,    y,    z <= 1
%
% where f(u) = exp(-u) and g(u) = 2 u^2 - 4 u, for all real u. It
% formulates and solves a simpler LP model by approximating f and
% g with piecewise-linear functions. Then it transforms the model
% into a MIP by negating the approximation for f, which corresponds
% to a non-convex piecewise-linear function, and solves it again.

names = {'x'; 'y'; 'z'};

try
    clear model;
    model.A = sparse([1 2 3; 1 1 0]);
    model.obj = [0; -1; 0];
    model.rhs = [4; 1];
    model.sense = '<>';
    model.vtype = 'C';
    model.lb = [0; 0; 0];
    model.ub = [1; 1; 1];
    model.varnames = names;

    % Compute f and g on 101 points in [0,1]
    u = linspace(0.0, 1.0, 101);
    f = exp(-u);
    g = 2*u.^2 - 4*u;

    % Set piecewise linear objective f(x)
    model.pwlobj(1).var = 1;
    model.pwlobj(1).x   = u;
    model.pwlobj(1).y   = f;

    % Set piecewise linear objective g(z)
    model.pwlobj(2).var   = 3;
    model.pwlobj(2).x     = u;
    model.pwlobj(2).y     = g;

    % Optimize model as LP
```

```
    result = gurobi(model);

    disp(result);

    for v=1:length(names)
        fprintf('%s %d\n', names{v}, result.x(v));
    end

    fprintf('Obj: %e\n', result.objval);

    % Negate piecewise-linear objective function for x
    f = -f;
    model.pwlobj(1).y = f;

    gurobi_write(model, 'pwl.lp')

    % Optimize model as a MIP
    result = gurobi(model);

    disp(result);

    for v=1:length(names)
        fprintf('%s %d\n', names{v}, result.x(v));
    end

    fprintf('Obj: %e\n', result.objval);

catch gurobiError
     fprintf('Error reported\n');
end
```

qcp.m

```
% Copyright 2017, Gurobi Optimization, Inc.

% This example formulates and solves the following simple QCP model:
%  maximize
%      x
%  subject to
%      x + y + z = 1
%      x^2 + y^2 <= z^2 (second-order cone)
%      x^2 <= yz        (rotated second-order cone)

clear model
names = {'x', 'y', 'z'};
model.varnames = names;

% Set objective: x
model.obj = [ 1 0 0 ];
model.modelsense = 'max';

% Add constraint: x + y + z = 1
model.A   = sparse([1 1 1]);
model.rhs = 1;
model.sense = '=';

% Add second-order cone: x^2 + y^2 <= z^2
model.quadcon(1).Qc = sparse([ 1 0  0;
                               0 1  0;
                               0 0 -1]);
model.quadcon(1).q  = zeros(3,1);
model.quadcon(1).rhs = 0.0;

% Add rotated cone: x^2 <= yz
model.quadcon(2).Qc = sparse([ 1 0  0;
                               0 0 -1;
                               0 0  0]);
model.quadcon(2).q  = zeros(3,1);
model.quadcon(2).rhs = 0;

gurobi_write(model, 'qcp.lp');

result = gurobi(model);

for j=1:3
    fprintf('%s %e\n', names{j}, result.x(j))
end
```

```
fprintf('Obj: %e\n', result.objval);
```

qp.m

```
% Copyright 2017, Gurobi Optimization, Inc.

% This example formulates and solves the following simple QP model:
%  minimize
%      x^2 + x*y + y^2 + y*z + z^2 + 2 x
%  subject to
%      x + 2 y + 3 z >= 4
%      x +   y       >= 1
%
% It solves it once as a continuous model, and once as an integer
% model.


clear model;
names = {'x', 'y', 'z'};
model.varnames = names;
model.Q = sparse([1 0.5 0; 0.5 1 0.5; 0 0.5 1]);
model.A = sparse([1 2 3; 1 1 0]);
model.obj = [2 0 0];
model.rhs = [4 1];
model.sense = '>';

gurobi_write(model, 'qp.lp');

results = gurobi(model);

for v=1:length(names)
    fprintf('%s %e\n', names{v}, results.x(v));
end

fprintf('Obj: %e\n', results.objval);

model.vtype = 'B';

results  = gurobi(model);

for v=1:length(names)
    fprintf('%s %e\n', names{v}, results.x(v));
end

fprintf('Obj: %e\n', results.objval);
```

sos.m

```
% Copyright 2017, Gurobi Optimization, Inc.

% This example creates a very simple Special Ordered Set (SOS)
% model. The model consists of 3 continuous variables, no linear
% constraints, and a pair of SOS constraints of type 1.

try
    clear model;
    model.ub = [1 1 2];
    model.obj = [2 1 1];
    model.modelsense = 'Max';
    model.A = sparse(1,3);
    model.rhs = 0;
    model.sense = '=';

    % Add first SOS: x1 = 0 or x2 = 0
    model.sos(1).type   = 1;
    model.sos(1).index  = [1 2];
    model.sos(1).weight = [1 2];

    % Add second SOS: x1 = 0 or x3 = 0
    model.sos(2).type   = 1;
    model.sos(2).index  = [1 3];
    model.sos(2).weight = [1 2];

    % Write model to file
    gurobi_write(model, 'sos.lp');

    result = gurobi(model);

    for i=1:3
        fprintf('x%d %e\n', i, result.x(i))
    end

    fprintf('Obj: %e\n', result.objval);

catch gurobiError
    fprintf('Encountered an error\n')
end
```

3.8 R Examples

This section includes source code for all of the Gurobi R examples. The same source code can be found in the `examples/R` directory of the Gurobi distribution.

lp.R

```
# Copyright 2017, Gurobi Optimization, Inc.
#
# This example formulates and solves the following simple LP model:
#  maximize
#        x + 2 y + 3 z
#  subject to
#        x +   y       <= 1
#              y +   z <= 1

library("Matrix")
library("gurobi")

model <- list()

model$A          <- matrix(c(1,1,0,0,1,1), nrow=2, byrow=T)
model$obj        <- c(1,1,2)
model$modelsense <- "max"
model$rhs        <- c(1,1)
model$sense      <- c('<=', '<=')

result <- gurobi(model)

print(result$objval)
print(result$x)

# Second option for A - as a sparseMatrix (using the Matrix package)...

model$A <- spMatrix(2, 3, c(1, 1, 2, 2), c(1, 2, 2, 3), c(1, 1, 1, 1))

params <- list(Method=2, TimeLimit=100)

result <- gurobi(model, params)

print(result$objval)
print(result$x)

# Third option for A - as a sparse triplet matrix (using the slam package)...

model$A <- simple_triplet_matrix(c(1, 1, 2, 2), c(1, 2, 2, 3), c(1, 1, 1, 1))
```

```
params <- list(Method=2, TimeLimit=100)

result <- gurobi(model, params)

print(result$objval)
print(result$x)
```

lp2.R

```
# Copyright 2017, Gurobi Optimization, Inc.
#
# Formulate a simple linear program, solve it, and then solve it
# again using the optimal basis.

library("gurobi")

model <- list()

model$A     <- matrix(c(1,3,4,8,2,3), nrow=2, byrow=T)
model$obj   <- c(1,2,3)
model$rhs   <- c(4,7)
model$sense <- c('>=', '>=')

# First solve - requires a few simplex iterations

result <- gurobi(model)

model$vbasis <- result$vbasis
model$cbasis <- result$cbasis

# Second solve - start from optimal basis, so no iterations

result <- gurobi(model)
```

mip.R

```
# Copyright 2017, Gurobi Optimization, Inc.
#
# This example formulates and solves the following simple MIP model:
#  maximize
#        x +   y + 2 z
#  subject to
#        x + 2 y + 3 z <= 4
#        x +   y       >= 1
#        x, y, z binary

library("gurobi")

model <- list()

model$A          <- matrix(c(1,2,3,1,1,0), nrow=2, ncol=3, byrow=T)
model$obj        <- c(1,1,2)
model$modelsense <- "max"
model$rhs        <- c(4,1)
model$sense      <- c('<=', '>=')
model$vtype      <- 'B'

params <- list(OutputFlag=0)

result <- gurobi(model, params)

print('Solution:')
print(result$objval)
print(result$x)
```

piecewise.R

```
# Copyright 2017, Gurobi Optimization, Inc.
#
# This example considers the following separable, convex problem:
#
#  minimize
#        f(x) - y + g(z)
#  subject to
#        x + 2 y + 3 z   <= 4
#        x +   y         >= 1
#        x,    y,    z   <= 1
#
# where f(u) = exp(-u) and g(u) = 2 u^2 - 4u, for all real u.  It
# formulates and solves a simpler LP model by approximating f and
# g with piecewise-linear functions.  Then it transforms the model
# into a MIP by negating the approximation for f, which gives
# a non-convex piecewise-linear function, and solves it again.

library("gurobi")

model <- list()

model$A     <- matrix(c(1,2,3,1,1,0), nrow=2, byrow=T)
model$obj   <- c(0,-1,0)
model$ub    <- c(1,1,1)
model$rhs   <- c(4,1)
model$sense <- c('<=', '>=')

# Uniformly spaced points in [0.0, 1.0]
u <- seq(from=0, to=1, by=0.01)

# First piecewise-linear function: f(x) = exp(-x)
pwl1     <- list()
pwl1$var <- 1
pwl1$x   <- u
pwl1$y   <- sapply(u, function(x) exp(-x))

# Second piecewise-linear function: g(z) = 2 z^2 - 4 z
pwl2     <- list()
pwl2$var <- 3
pwl2$x   <- u
pwl2$y   <- sapply(u, function(z) 2 * z * z - 4 * z)

model$pwlobj <- list(pwl1, pwl2)
```

```
result <- gurobi(model)

print(result$objval)
print(result$x)


# Negate piecewise-linear function on x, making it non-convex

model$pwlobj[[1]]$y <- sapply(u, function(x) -exp(-x))

result <- gurobi(model)
gurobi_write(model, "junk.lp")

print(result$objval)
print(result$x)
```

qcp.R

```
# Copyright 2017, Gurobi Optimization, Inc.
#
# This example formulates and solves the following simple QCP model:
#  maximize
#        x
#  subject to
#        x + y + z   =  1
#        x^2 + y^2 <= z^2  (second-order cone)
#        x^2 <= yz         (rotated second-order cone)

library("gurobi")
library("Matrix")

model <- list()

model$A          <- matrix(c(1,1,1), nrow=1, byrow=T)
model$modelsense <- "max"
model$obj        <- c(1,0,0)
model$rhs        <- c(1)
model$sense      <- c('=')

# First quadratic constraint: x^2 + y^2 - z^2 <= 0
qc1 <- list()
qc1$Qc <- spMatrix(3, 3, c(1, 2, 3), c(1, 2, 3), c(1.0, 1.0, -1.0))
qc1$rhs <- 0.0

# Second quadratic constraint: x^2 - yz <= 0
qc2 <- list()
qc2$Qc <- spMatrix(3, 3, c(1, 2), c(1, 3), c(1.0, -1.0))
qc2$rhs <- 0.0

model$quadcon <- list(qc1, qc2)

result <- gurobi(model)

print(result$objval)
print(result$x)
```

qp.R

```
# Copyright 2017, Gurobi Optimization, Inc.
#
# This example formulates and solves the following simple QP model:
#  minimize
#        x^2 + x*y + y^2 + y*z + z^2 + 2 x
#  subject to
#        x + 2 y + 3z >= 4
#        x +   y      >= 1

library("gurobi")

model <- list()

model$A     <- matrix(c(1,2,3,1,1,0), nrow=2, byrow=T)
model$Q     <- matrix(c(2,1,0,1,2,1,0,1,2), nrow=3, byrow=T)
model$obj   <- c(2,0,0)
model$rhs   <- c(4,1)
model$sense <- c('>=', '>=')

result <- gurobi(model)

print(result$objval)
print(result$x)
```

sos.R

```
# Copyright 2017, Gurobi Optimization, Inc.
#
# This example formulates and solves the following simple SOS model:
#  maximize
#        2 x + y + z
#  subject to
#        x1 = 0 or x2 = 0 (SOS1 constraint)
#        x1 = 0 or x3 = 0 (SOS1 constraint)
#        x1 <= 1, x2 <= 1, x3 <= 2

library("gurobi")

model <- list()

model$A          <- matrix(c(0,0,0), nrow=1, byrow=T)
model$obj        <- c(2,1,1)
model$modelsense <- "max"
model$ub         <- c(1,1,2)
model$rhs        <- c(0)
model$sense      <- c('=')

# First SOS: x1 = 0 or x2 = 0
sos1 <- list()
sos1$type <- 1
sos1$index <- c(1, 2)
sos1$weight <- c(1, 2)

# Second SOS: x1 = 0 or x3 = 0
sos2 <- list()
sos2$type <- 1
sos2$index <- c(1, 3)
sos2$weight <- c(1, 2)

model$sos <- list(sos1, sos2)

result <- gurobi(model)

print(result$objval)
print(result$x)
```